PP cheat sheet

Bitni algoritmi, formule i složenosti iz skripte:

---

1. Amdahlov zakon

- **Formula**:

$$\text{Ubrzanje} = \frac{1}{1-p}$$

  gdje je p paralelizirani dio programa.
- **Proširena formula** (s brojem procesora N_p):

$$\text{Ubrzanje} = \frac{1}{s + \frac{p}{N_p}}$$

  gdje je s sekvencijalni dio, a p paralelni dio.

---

2. Gustafsonov zakon

- **Formula**:

$$\text{Ubrzanje} = N_p + (1 - N_p) \cdot s$$

  gdje je N_p broj procesora, a s sekvencijalni dio.

---

3. Algoritam zbroja prefiksa (prefix sum)

- **Opis**: Paralelno izračunava kumulativne vrijednosti niza koristeći asocijativni operator.
- **Složenost**:
  - Redukcija: $O(\log n)$ koraka.
  - Down-sweep: $O(\log n)$ koraka.
  - Ukupno: $O(\log n)$ s $O(n)$ procesora (PRAM).
- **Primjene**: Sortiranje, računalna geometrija, alokacija procesora.

---

4. Redukcija (MPI_Reduce)

- **Složenost**:

- Korištenjem binarnog stabla: $O(\log n)$.
- U asinkronom PRAM modelu: $O\left(\frac{B \log n}{\log B}\right)$, gdje je B trošak sinkronizacije.

---

5. Monte Carlo metoda za računanje π (MPI primjer)

- **Algoritam**: Paralelno generiranje slučajnih točaka i redukcija rezultata.
- **MPI funkcije**: `MPI_Bcast`, `MPI_Reduce`.
- **Složenost**: Ovisi o broju procesora N_p, s linearnim ubrzanjem.

---

6. Metoda konačnih razlika (Jacobi i Gauss-Seidel)

- **Jacobi**:
  - Paralelizacija bez ovisnosti o redoslijedu (svi elementi ažuriraju se istovremeno).
- **Gauss-Seidel**:
  - Teže za paralelizaciju zbog ovisnosti o redoslijedu.
- **Složenost**: Ovisi o broju iteracija i komunikacijskoj strukturi (npr. 2D mreža).

---

7. Algoritam hiperkocke (butterfly)

- **Opis**: Koristi se za globalne operacije poput redukcije ili broadcasta.
- **Složenost**:
  - Za n procesora: $O(\log n)$ koraka.
  - Primjer: Zbrajanje svih elemenata niza.

---

8. Brentovo pravilo

- **Složenost**:

$$O\left(\frac{n}{p} + \log p\right)$$

gdje je n veličina problema, a p broj procesora.

---

9. Algoritam za pretraživanje stabla (branch-and-bound)

- **Strategije**:

- Dinamičko raspoređivanje zadataka (voditelj/radnik).
- Složenost: Ovisi o dubini stabla i učinkovitosti grananja.

---

10. PRAM modeli

- **EREW PRAM**:
  - Složenost operacija (npr. redukcija): $O(\log n)$.
- **CRCW PRAM**:
  - Konkurentno pisanje s slučajnim odabirom pobjednika.

---

11. Asinkroni PRAM (aPRAM)

- **Složenost redukcije**:

$$O\left(\frac{B \log n}{\log B}\right)$$

  gdje je B trošak sinkronizacije.

---

12. Algoritmi za ujednačavanje opterećenja

- **Cikličko pridruživanje**: Ravnomjerna distribucija zadataka.
- **Vjerojatnosne metode**: Slučajna distribucija za velike skupove zadataka.

Ključni koncepti:

- **Zrnatost (granularnost)**: Omjer računanja i komunikacije (sitnozrnati vs. krupnozrnati).
- **Lokalnost**: Minimiziranje pristupa udaljenoj memoriji.
- **Sinkronizacija**: Korištenje `MPI_Barrier` ili implicitnih ograda u komunikaciji.

Evo pseudokoda za algoritme **reduciranje** (redukcija) i **down_sweep** korištene u operaciji zbroja prefiksa (prefix sum) iz skripte:

---

1. Algoritam +_reduciranje

Cilj: Izračunaj ukupni zbroj elemenata niza koristeći binarno stablo.
Ulaz: Niz `A[]` duljine `n` (pretpostavka: `n` je potencija broja 2).
Izlaz: Ukupni zbroj u `A[n-1]`.

```
algoritam +_reduciranje(A[], n):
    for d = 0 to log2(n) - 1:
        paralelno za svaki i od 0 do n-1 korakom 2^(d+1):
            lijevo = i + 2^d - 1     # Indeks lijevog djeteta
            desno = i + 2^(d+1) - 1  # Indeks desnog djeteta
            A[desno] += A[lijevo]
```

Primjer:
Za `n = 8`, postupak redukcije izgleda ovako:

```
Korak d=0: Kombinira elemente udaljene za 1 (npr. A[1] += A[0], A[3] += A[2], ...)
Korak d=1: Kombinira elemente udaljene za 2 (npr. A[3] += A[1], A[7] += A[5], ...)
Korak d=2: Kombinira elemente udaljene za 4 (npr. A[7] += A[3])
```

---

2. Algoritam down_sweep

Cilj: Nakon redukcije, izračunaj sve prefiksne zbrojeve.
Ulaz: Niz `A[]` s ukupnim zbrojem u `A[n-1]`.
Izlaz: Prefiksni zbrojevi u `A[0..n-1]`.

```
algoritam down_sweep(A[], n):
    A[n-1] = 0  # Postavi korijen stabla na neutralni element (0 za zbrajanje)
    for d = log2(n) - 1 downto 0:
        paralelno za svaki i od 0 do n-1 korakom 2^(d+1):
            lijevo = i + 2^d - 1
            desno = i + 2^(d+1) - 1
            temp = A[lijevo]
            A[lijevo] = A[desno]       # Prenesi vrijednost s desnog na lijevo
dijete
            A[desno] = temp + A[desno] # Ažuriraj desno dijete
```

Primjer:
Za `n = 8`, nakon redukcije:

```
Korak d=2: A[3] = 0, A[7] = A[3] + stara vrijednost
Korak d=1: Ažurira parove udaljene za 2 (npr. A[1], A[3])
Korak d=0: Ažurira parove udaljene za 1 (npr. A[0], A[1])
```

---

3. Kombinacija za +_prescan

Algoritam za prefiksne zbrojeve koristi oba koraka:

```
algoritam +_prescan(A[], n):
    +_reduciranje(A[], n)  # Izračunaj ukupni zbroj
    down_sweep(A[], n)     # Izračunaj prefiksne zbrojeve
```

Svojstva:

- **Složenost**: Oba algoritma imaju složenost **O(log n)** koraka uz **O(n)** procesora (PRAM model).
- **Asocijativnost**: Zahtijeva se da operator (npr. `+`) bude **asocijativan**.
- **Primjene**: Sortiranje, paralelno ažuriranje polja, algoritmi nad stabilima.

Ilustracija (za `n=8`):

```
Redukcija:          Down-sweep:
7 → 15              15 → 0 → 7 → 15
3 → 7   11 → 15     7  → 0   15 → 11
1 3 5 7 9 11 13 15  1  3  5  7  9 11 13 15
```

Evo pseudokoda za **prescan** (prefiksni zbroj bez trenutnog elementa) i **scan** (klasični prefiksni zbroj) algoritme iz skripte:

---

1. Prescan (prefiksni zbroj *bez* trenutnog elementa)

Cilj: Izračunaj $[0, a_0, a_0+a_1, ..., a_0+a_1+...+a_{n-2}]$.
Koraci: Redukcija + propagacija vrijednosti kroz stablo (*down-sweep*).

```
algoritam prescan(A[], n):
    # Korak 1: Redukcija (izračunaj ukupni zbroj)
    for d = 0 to log2(n) - 1:
        paralelno za svaki i od 0 do n-1 korakom 2^(d+1):
            lijevo = i + 2^d - 1
            desno = i + 2^(d+1) - 1
            A[desno] += A[lijevo]

    # Spremi ukupni zbroj prije nego što ga postaviš na 0
    total = A[n-1]

    # Korak 2: Down-sweep (propagacija vrijednosti)
    A[n-1] = 0  # Postavi korijen na neutralni element (0 za zbrajanje)
```

```
    for d = log2(n) - 1 downto 0:
        paralelno za svaki i od 0 do n-1 korakom 2^(d+1):
            lijevo = i + 2^d - 1
            desno = i + 2^(d+1) - 1
            temp = A[lijevo]
            A[lijevo] = A[desno]
            A[desno] = temp + A[desno]

    return total  # Vrati ukupni zbroj za kasniju upotrebu
```

---

2. Scan (klasični prefiksni zbroj)

Cilj: Izračunaj $[a_0, a_0+a_1, \ldots, a_0+a_1+\ldots+a_{n-1}]$.
Koraci: Prescan + pomak rezultata i dodavanje ukupnog zbroja.

```
algoritam scan(A[], n):
    # Korak 1: Izračunaj prescan i spremi ukupni zbroj
    total = prescan(A[], n)

    # Korak 2: Pomakni prescan za 1 mjesto ulijevo i dodaj ukupni zbroj na kraj
    paralelno za svaki i od 0 do n-1:
        if i == 0:
            novi_A[i] = A[i]  # Prvi element ostaje 0 (prescan)
        else:
            novi_A[i] = A[i] + originalni_A[i-1]  # Pretpostavka da imamo kopiju originalnog niza

    # Alternativno (bez kopije originalnog niza, ali zahtijeva dodatnu memoriju):
    # 1. Stvori privremeni niz za pohranu prescan vrijednosti
    # 2. Pomakni vrijednosti ulijevo i dodaj originalne elemente
```

Primjer za `n = 4`:

Ulaz: `A = [3, 1, 7, 0]`
Prescan:

1. Nakon redukcije: `A = [3, 4, 7, 11]`
2. Nakon down-sweep: `A = [0, 3, 4, 11]`
   Scan:

- Pomakni prescan za 1 ulijevo: `[3, 4, 11, 0]`
- Dodaj originalne elemente: `[3, 3+1, 4+7, 11+0] = [3, 4, 11, 11]`

Napomene:

1. **Složenost**: Oba algoritma imaju složenost **O(log n)** koraka uz **O(n)** procesora (PRAM model).
2. **Asocijativnost**: Operator (npr. `+`, `max`, `min`) mora biti asocijativan.
3. **Primjene**:
   - **Prescan**: Alokacija memorije, radix sort, paralelno grananje.
   - **Scan**: Kumulativni zbrojevi, sortiranje, obrada signala.