Binomial Heaps si Fibonacci Heaps

Binomial Heaps **Ce sunt:**
Un binomial heap este o colectie de arbori binomiali care respecta proprietatea de heap (min-heap sau max-heap). Este folosit in implementarea cozilor de prioritati. **Arbori binomiali:**
- B0: un singur nod.
- Bk: se obtine atasand un arbore B(k-1) ca fiu stanga al radacinii altui arbore B(k-1).
- Proprietati: Bk are 2^k noduri, inaltime k, radacina are exact k copii.

Structura heap-ului:
- Heap-ul este o lista de arbori binomiali ordonati dupa grad.
- Exista cel mult un arbore binomial pentru fiecare grad.
- Structura seamana cu reprezentarea binara a unui numar (daca ai n noduri, structura arborilor corespunde reprezentarii binare a lui n).

Operatii:
- Find-Min: cauti radacina minima din lista. Complexitate O(log n).
- Union: unesti doua heap-uri, la fel ca adunarea binara (cand doi arbori au acelasi grad se combina). Complexitate O(log n).
- Insert(x): creezi un heap cu un singur nod si faci Union. Amortizat O(1), worst-case O(log n).
- Extract-Min: gasesti arborele cu radacina minima, il elimini si unesti copiii radacinii cu restul heap-ului. O(log n).
- Decrease-Key(x): scazi valoarea unui nod si il deplasezi in sus pentru a pastra proprietatea heap. O(log n).
- Delete(x): faci Decrease-Key(x, -inf), apoi Extract-Min. O(log n).

Complexitati:
Find-Min: O(log n)
Insert: O(1) amortizat, O(log n) worst-case
Union: O(log n)
Extract-Min: O(log n)
Decrease-Key: O(log n)
Delete: O(log n)

Unde se folosesc:
- In algoritmi pe grafuri (Dijkstra, Prim) cand sunt necesare multe operatii Union.

Avantaje: bune pentru multe operatii de tip Union.
Dezavantaje: mai greu de implementat si mai lent decat Fibonacci heaps in teorie.

Fibonacci Heaps **Ce sunt:**
Un fibonacci heap este o structura de coada cu prioritati, generalizare a binomial heap. Pastreaza o colectie de arbori intr-o lista circulara dublu inlantuita a radacinilor, cu un pointer catre minim.

Structura:
- Colectie de arbori, nu neaparat binomiali.
- Lista circulara a radacinilor.
- Pointer direct la elementul minim.

Operatii:
- Find-Min: O(1) - avem pointer direct la minim.
- Insert(x): adauga un nod in lista de radacini si actualizeaza minimul. O(1) amortizat.
- Union(H1,H2): concateneaza listele de radacini si stabileste minimul. O(1).
- Extract-Min: elimina nodul minim, muta copiii sai in lista de radacini, apoi consolideaza arborii de acelasi grad. O(log n) amortizat.
- Decrease-Key(x,k): scade valoarea lui x; daca incalca proprietatea heap, nodul se taie si se muta in lista de radacini (cascading cut daca e nevoie). O(1) amortizat.
- Delete(x): se face Decrease-Key(x,-inf) si apoi Extract-Min. O(log n) amortizat.

Complexitati (amortizate):
Find-Min: O(1)
Insert: O(1)
Union: O(1)

Decrease-Key: O(1)
Delete: O(log n)
Extract-Min: O(log n)
Unde se folosesc:
- In algoritmi de grafuri intensivi in Decrease-Key, ca Dijkstra sau Prim.
Avantaje:
- Insert si Decrease-Key extrem de rapide.
- Foarte bune pentru algoritmi unde se fac multe astfel de operatii.
Dezavantaje:
- Implementare complexa.
- In practica, constantii pot face un binary heap mai rapid pe date reale.