Giorno 6
Garbage Collection

1 Garbage Collection

[tratto solo gli algoritmi, si vedano le prime slide per overview di allocazione in heap motivazioni, definizione di frammentazione interna ed esterna]

1.1 Reference Counting

Per ogni cella si mantiene un contatore dei riferimenti a quella cella. Quando questo arriva a zero, la cella diventa garbage e si può liberare.

Problema: potrei rimanere con una **componente connessa** di celle che riferiscono le une alle altre che è irraggiungibile;

1.2 Mark-Sweep

Faccio una visita del grafo delle reference e "marco" tutto ciò che è raggiungibile. Il resto è garbage;

In questo modo risolvo il problema delle componenti connesse irraggiungibili.

Problemi: Come si identificano i puntatori? Si deve avere informazioni sul **tipo**.

- Java: non è un problema, si hanno informazioni e sappiamo "chi è una reference"
- OCaml: Si sacrifica il bit meno significativo, che è 0 se e solo se il dato è un puntatore (lo sarebbe comunque, poiché i puntatori sono allineati alle parole, ma si pone uguale a 1 se non si ha un puntatore).

Altri problemi: Bisogna sospendere l'esecuzione; Non si interviene sulla **frammentazione**.

1.3 Copying collection - Algoritmo di Cheney

Questo metodo risolve il problema della frammentazione, aggiungendo un po' di overhead e sacrificando metà della memoria.

- Si suddivide lo heap in due parti uguali, dette "from-space" e "to-space"
- Si può allocare solo nel **from-space**
- Periodicamente si copiano tutte le celle raggiungibili nel **to-space**, deframmentandole
- Si scambiano i ruoli di from-space e to-space ad ogni copia; in questo modo abbiamo effettivamente **perso ogni riferimento irraggiungibile**.

1.4 GC Generazionale

Si suddivide lo heap in n aree, dove l'area 0 è detta `young`, l'area $n-1$ è detta `old` e le aree in mezzo sono "generazioni intermedie".

Inizialmente tutti gli oggetti sono creati nell'area `young`, e periodicamente gli oggetti young raggiungibili sono spostati in un'area "più old". Gli oggetti `young` sono ripuliti più spesso, mentre quelli old meno spesso (ma sono di più, quindi ha costo maggiore);

Questo è l'approccio di Java e C#, che hanno entrambi tre generazioni, ognuna che **utilizza un algoritmo diverso** per la garbage collection, e.g. Java usa **copy-collection** per le generazioni 0,1 e mark-sweep (+ meccanismi anti frammentazione) per la generazione 2.