Tema NR 4: Hashtabeller

Emma Persson `empe5691`
Elise Edette `tero0337`
Aframyeos Rohoum `afro0793`

12 februari 2015

1 Förslag på muntafrågor:

1. Förklara begreppet kollission. VG: Vilka metoder finns det för att undvika kollission?

2. Förklara skillnaden mellan linjär probing och kvadratisk probing. VG: Vilka risker/problem kan uppstå med kvadratisk probing när arrayen börjar bli halvfull?

2 Hashtabeller och hashnig:

2.1 Hashtabeller

En hashtabell är ett sätt att lagra data med hjälp av en nyckel, ofta i en array. Nyckel görs om till ett index med hjälp av

hashing. Hashtbeller används ofta i de fall då det är viktigt att insättning av element och sökning går snabbt. Till skillnad från i t.ex en länkad lista, där varje element måste gås igenom för att hitta den plats där det element man söker efter finns, kan man i en hashtabell snabbt hitta det element man söker efter genom att direkt gå till den plats i arrayen där man vet att det ligger. Vid insättning i en hashtabell, skapas med hjälp av en hashfunktion en kod, som motsvarar ett index i en array där elementet stoppas in. När en sökning efter ett element sedan ska göras, räknas samma kod fram igen, till indexet i arrayen där elementet i bästa fall finns.

2.2 Hashing

Hashkoden skapas med en hashfunktion. Hashfunktionen räknar om delar av elementets data till en kod, detta kallas för hashing. En hashfunktionen ska alltid ge samma kod för samma objekt. En mindre effektiv hashkod gör ofta att flera element hamnar på samma plats, medan en mer effektiv hashfunktion gör att elementen sprids ut jämt över hela arrayen, och därmed minimerar risken för att flera element ska få samma kod.

2.3 Kollissionshantering

Hashning ska ge en kod som är unik i största möjliga mån, så att så få kollisioner som möjligt uppstår. I figur ett kan man se en kollision när element 64 ska sättas in på index 4. Hashfunktionen, som i detta fall är f(x)= x % 10 har alltså resulterat i en kollision. Det finns flera olika sätt att hantera kollisioner,

som fungerar olika bra vid olika tillfällen. Ett sätt att hantera kollisioner är linear probing. .

11 mod 10 = 1. 44 mod 10 = 4. 64 mod 10 = 4

0	
1	11
2	
3	
4	44 64
5	
6	56
7	
8	
9	

Figur 1. Kollission

2.3.1 Linear probing

Linear probing innebär att ett element vid en kollision placeras på den första lediga platsen under det tänkta indexet. Detta kan lätt skapa problem, då flera av de efterföljande platserna redan kan vara upptagna av andra element, och den första lediga plat-

sen i värsta fall kan vara i slutet på arrayen. Det är vanligt att kollisionshantering med linear probing skapar så kallad ”clustering”, vilket innebär att många element hamnar i kluster, och inte får en jämn spridning över hela arrayen. Då element ska sökas upp ur hashtabellen, måste en jämförelse göras med alla element på och under det efterfrågade indexet, tills elementet hittas, vilket kan ta lång tid.

2.4 Quadratic probing

Quadratic probing är en annan metod för att hantera kollisioner. Quadratic probing liknar linear probing, men med den skillnad att då en kollision uppstår, stoppas elementet inte in på nästa lediga plats, utan en ny plats räknas ut genom att ta det ursprungliga indexet i kvadrat. För att vara säker på att den nya platsens index finns i arrayen, görs en modulusberäkning på det nya indexet, och modulus med arrayens storlek. Quadratic probing fungerar bäst med en array där storleken är ett primtal. Detta för att de nya indexen vid kollision med större sannolikhet blir unika. Tabellen bör inte låtas bli mer än halvfull, eftersom det då inte är säkert att det går att hitta en ledig plats åt element som kolliderar och behöver en ny plats. Denna metod minskar risken för clustering, och sprider ut elementen mer i arrayen.

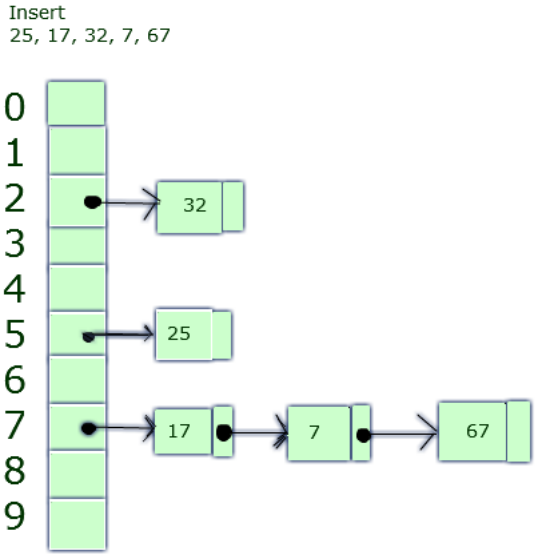
2.5 Double hashing

Double hashing är ytterligare en metod för att hantera kollisioner. Double hashing innebär att en hashkod tas fram utifrån

elementet som ska placeras. Om en kollision uppstår på denna plats, räknas en ny hashkod ut, för det intervall som ska hoppas fram i arrayen. På denna nya plats kan sedan elementet stoppas om, om den är ledig. Med denna metod blir risken för clustering ännu lägre än vid linjär probing och quadratic probing.

2.6 Separate chaining

Ett annat sätt att hantera kollisioner är att tillämpa separate chaining (se figur 2). Detta innebär att varje element i arrayen är en referens till en länkad lista. En länkad lista är en typ av lista där varje element innehåller sin egen data, och en referens till nästa element i listan. Om flera element råkar få samma hashkod, stoppas de in längst fram i den länkade listan. En sökning efter ett element innebär då att platsen i arrayen först hittas med hjälp av hashkoden, varpå platsens länkade lista söks igenom efter det önskade elementet. En nackdel med detta sätt att hantera kollisioner är att då flera element har kolliderat, kan den länkade listan bli lång, vilket innebär att den tar lång tid att söka igenom. Ytterligare en nackdel med separate chaining är att de länkade listorna tar förhållandevis stor plats. Fördelen med separate chaining är att man kan lätt ta bort element genom att länka om listan, detta är inte möjligt med probing eller double hashing eftersom data kan hamna på en annan index än hashfunktionen ger.



Figur 2. Separate chaining