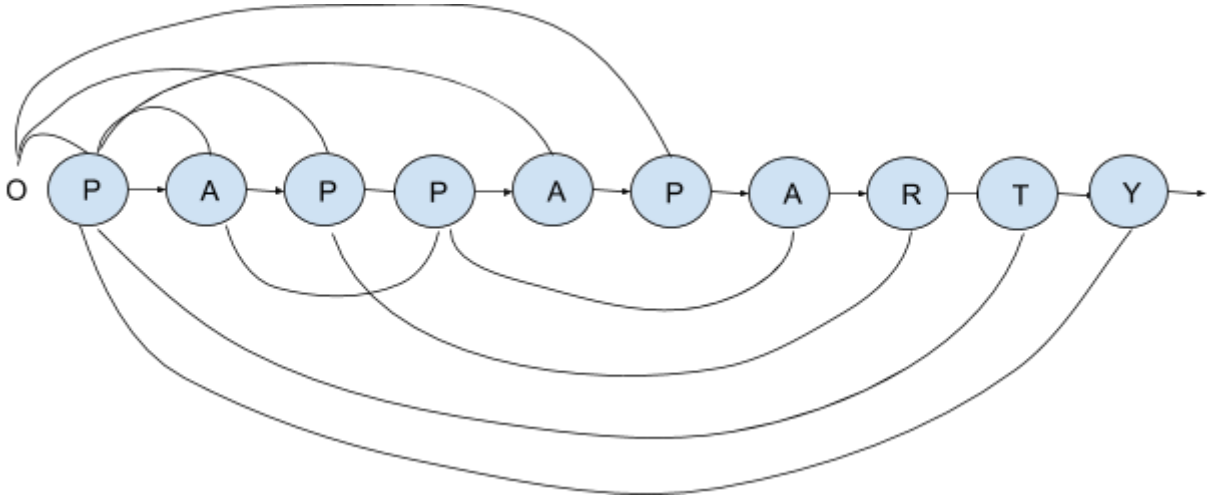
Tildatenta 2017-10-20 Lösningsskiss

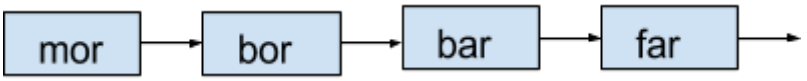
E-delen

1. **KMP PAPPAPARTY**



next[i] = 0 1 0 2 1 0 4 3 1 1

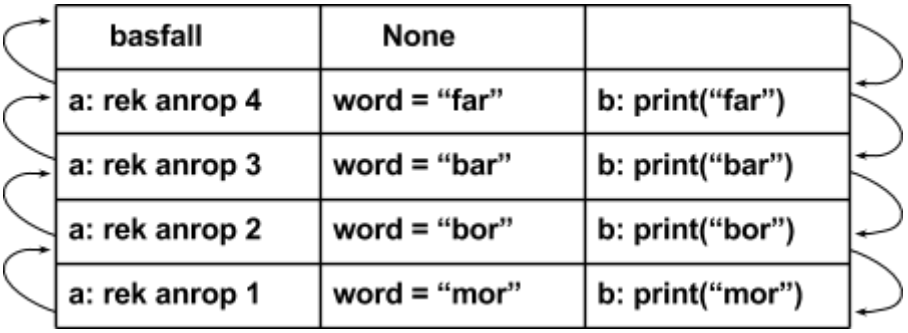
2. **Parent-pekare**



Utskriftfunktionen fungerar så här:

1. Om noden inte är None
    a. gör vi först ett rekursivt anrop med den nod som parent pekar på
    b. och sen skriver vi ut nodens ord

Vi anropar med första noden, gör ett rekursivt anrop, men sparar först på stacken. Samma sak för nästa osv... I fjärde rekursiva anropet är parent None, rekursionen avbryts. Sen poppas föregående anrop från stacken, print-satsen utförs, osv.



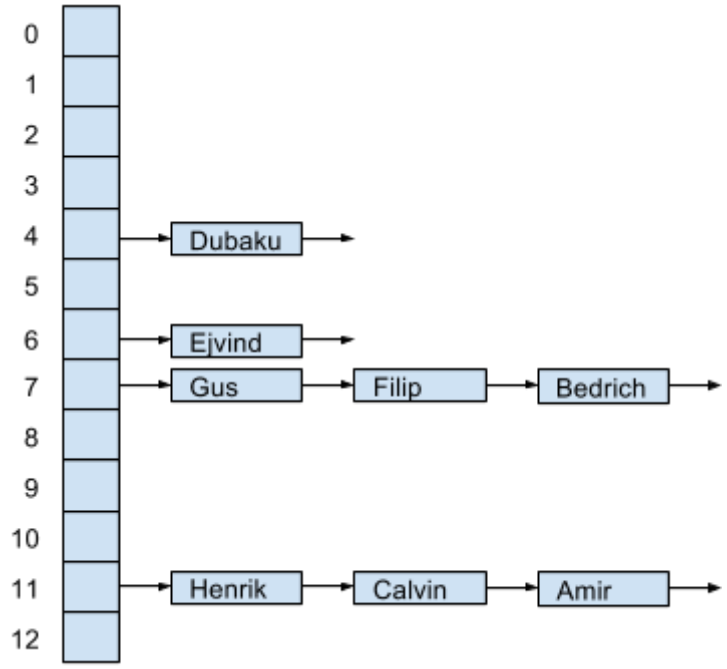
basfall	None	
a: rek anrop 4	word = “far”	b: print(“far”)
a: rek anrop 3	word = “bar”	b: print(“bar”)
a: rek anrop 2	word = “bor”	b: print(“bor”)
a: rek anrop 1	word = “mor”	b: print(“mor”)

3. **Kryptering**

   One-time pad: *viktigt* med säker nyckelöverföring. Bägge parter måste ha nyckeln (samma nyckel används vid kryptering och dekryptering).

   RSA: *inte viktigt* med säker nyckelöverföring. Den som krypterar meddelandet kan använda offentliga (publika) nyckeln, den som dekrypterar har sin egen privata nyckel, som inte behöver föras över till någon annan.

4. **Hasha pappor**



5. **Komplexitet**

   1. Vilken Ordo-klass?
      a. O(nlogn)
      b. O(n²)
      c. O(n)
      d. O(n²)
      e. O(logn)
      f. O(k^n)
   2. Rangordning: O(logn), O(n), O(nlogn), O(n²), O(k^n)
      ..dvs E C A B/D F

6. Syntax för faderskapstest

a) TH01 06,07 Godkänns inte, <Barn> och <Pappa> saknas/<Antal> start ej 0:a
b) D5S818 vWA CSF1PO TH01 Godkänns inte, för många <Locus>/<Mamma> etc saknas
c) CSF1PO 19,21 19,19 10,21 Godkänns
d) vWA 15,16 CSF1PO 14,14 TH01 7,9 Godkänns inte, <Locus> istället för <Barn>

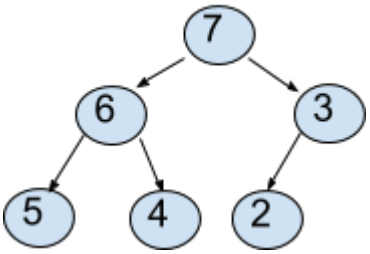
C-delen

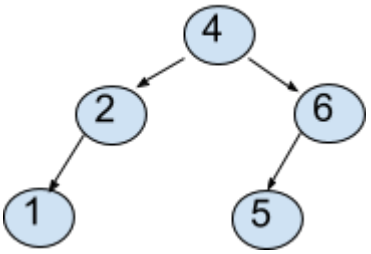
7. Komprimering

Många olika jämförelser möjliga.

egenskap	Huffmankodning	Lempel-Ziv
antal koder	Det finns m = 100 000 tecken (ca 2^{17}). Varje tecken får en egen binär kod (totalt m koder) på 17 bitar per tecken i medel. Om vissa tecken är vanligare får dessa kortare koder med Huffmankodning.	Tabellen kommer att innehålla inte bara de enskilda tecknen utan också vanliga teckenkombinationer, alltså får tabellen storleken m+x Om vissa följder av tecken förekommer flera gånger hamnar dessa i tabellen och får egna koder. Tabellen blir mycket stor. (Värsta fallet: Anta att vi blandar alfabeten, och att varje namn är k tecken långt. Totala antalet permutationer är m!/(m-k)!) Förslag: Begränsa längden på de teckenföljder som kan läggas in (maxlängd för ett namn).
tidskomplexitet = tiden för att 1. skapa koderna 2. komprimeringen: slå upp koden för varje tecken	m = antal tecken i skriftspråken n = antal tecken i namnlistan 1.Skapa koderna: - sorting (O(mlogm) - grenar O(mlogm) 2.Komprimeringen: - hashtabell O(m) - koda O(n) Totalt: O(mlogm) + O(n)	Vi går bara igenom texten en gång (O(n)) och komprimerar samtidigt som vi skapar tabellen. För varje tecken i namnen måste vi slå upp i tabellen, och ev. lägga in en ny kod. Bäst att i förväg skapa hashabell av storlek m+x.

8. Heap eller binärt sökträd





1. Lösning
    a. Se ovan till vänster. Hepvillkoret måste vara uppfyllt, dvs föräldern större än barnen (eller tvärtom för min-heap), för varje nod.
    b. Se ovan till höger. Mindre sorteras till vänster, större till höger, gäller för varje nod.
    c. Struktur och ordning, t ex något av följande:
        i. En heap måste vara komplett, det måste inte ett binärtäd
        ii. I en heap måste heapvillkoret vara uppfyllt för varje nod, binärträdet har mindre värden till vänster och större till höger.
        iii. En heap lagras i en array, ett binärträd har noder med left- och right-pekare
        iv. Även praktiska experiment duger som svar, t ex “prova att stoppa in ett nytt element”, “skriv ut trädet”
    d. Snabb sökning.
    e. Sortering (heapsort) eller prioritetskö eller bästaförstsökning.

A-delen

9. Lukes pappas problem

a) Algoritm

Indata: Darth:s startpunkt p_{start}
Rebellernas gömställe p_{slut}
Matrisen *connected* som anger vilka punkter som är förbundna med maskhål.
Utdata: Utskrift av en lista med punkter $p_{start}...p_{slut}$ som ger kortaste vägen

1. Skpa en datastruktur *used* för använda punkter (se b)
2. Skapa en kö
3. Lägg in en nod p_{start} med förälder None i kön
4. Markera p_{start} som True i *used*
5. Plocka ut en punkt **p** ur kön (med dequeue)
6. Gå igenom varje matriselement på p:s rad i matrisen *connected*:
    a. Om p_i inte redan är använd och om *connected*[**p**,p_i] är True:
        i. Lägg in en nod p_i med **p** som förälder kön
        ii. Lägg till p_i till datastrukturen med använda punkter
        iii. Avbryt om $p_i = p_{slut}$, skriv ut lösningen (enligt E-uppgift 2)
7. Upprepa från punkt 4 så länge kön inte är tom
8. Om kön blev tom, berätta för Darth att ingen väg finns (om du vågar)

b) Datastrukturer

A. Boolesk lista *used* med n platser, använd[i] sätts till True i 5.a.ii
B. Kö med noder
C. Noder med en punkt och en pappapekare
D. (och den givna matrisen connected)

c) Demonstrera hur algoritmen fungerar

d) Tidskomplexitet

Problemträdet blir aldrig större än de n punkterna
För varje punkt går vi igenom n-1 matriselement
Att kontrollera dumbarn beror på vald datastruktur, men max n
Totalt n*(n-1 + n) dvs $O(n^2)$

: