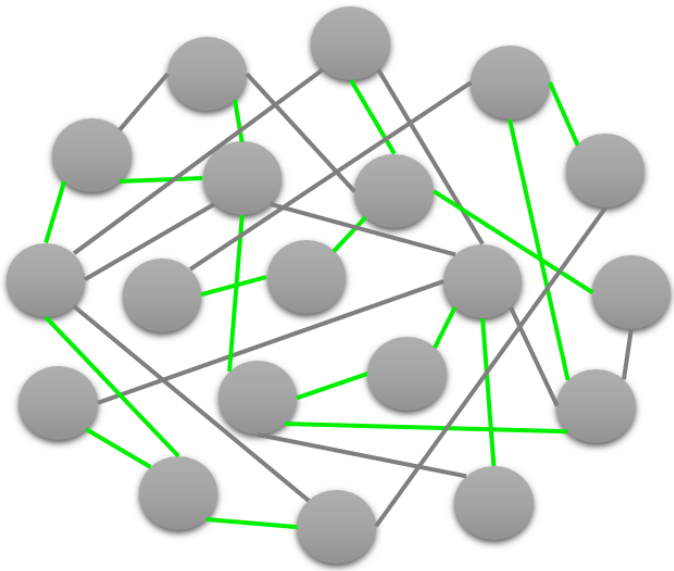
Structuri de date și algoritmi

There is no silver bullet

Fred Brooks

Structuri de date și algoritmi

APLICAȚII PRACTICE



Gabriel M. Danciu

[——DRAFT——]

Last Updated: 20 iulie 2017

2017

Editura Universitatii Transilvania

Publicat de:

Editura Universitatii Transilvania
Str. Iuliu Maniu nr. 41A Braşov
Brașov, 500091

Imprimată de Editura Universitatii Transilvania

Imprimată în România

Danciu, Gabriel, M.

Structuri de date și algoritmi:
aplicații practice/ Danciu, Gabriel, M.

p. cm.

Include referințe bibliografice și index

ISBN: 978-4-4444444-4-6 (format fizic)

ISBN: 978-5-5555555-5-7 (format electronic)

1. Get from QualityBooks

I. Danciu, Gabriel, M. , 2017 II. Title.

QP333.K33 2017

333.33-dc33 2017xxxxxx

Editura Universitatii Transilvania

Brașov România

Prima Ediție

14 13 12 11 10 / 10 9 8 7 6 5 4 3 2 1

Rezumatul cuprinsului

Cuprinsul detaliat

Lista de figuri

Prefață

Here you write a nice preface that makes people really want to read your book.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue.

Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

This is an illustration of the use of the HREF command. This should always be used since otherwise hyperlinks will not be properly formatted for print, PDF, and eBooks. – Google Scholar will help.

Sloboken, NJ
≈≈ October, 2016

1 Introducere

Pentru început ne vom familiariza cu principiile, notaţiile folosite de-a lungul întregului curs. Vom începe printr-o plasare a conceptului de algoritm în raport cu problemele reale utilizând un exemplu clasic. Vom prezenta conceptul de pseudocod și de asemenea vom folosi un exemplu de algoritm ușor de înteles. În finalul acestui capitol vom analiza algoritmii din punct de vedere a timpului de execuție al acestora.

Ce fel de probleme se rezolvă cu ajutorul algoritmilor? Să zicem că aproape orice aplicație dezvoltată conține un algoritm (nu neapărat în forma lui didactică). De la rețele de socializare, la comerț electronic, la industrie, în aproape orice domeniu legat mai mult sau mai puțin de știința calculatoarelor, vom găsi necesitatea implementării cel puțin a unui algoritm.

Primul aspect care trebuie definit este ce reprezintă un agloritm. Există în mod evident mai multe căi de a defini un algoritm și vom enumera în cele ce urmează câteva dintre acestea:

- Un algoritm este o procedură bine definită ce are ca intrare un set de valori sau una singură, și va produce una sau mai multe ieșiri.
- Un algoritm reprezintă un set de reguli definit pentru a rezolva o problemă într-un număr finit de pași.
- Un set de pași pentru a rezolva o problemă matematică sau pentru a realiza un proces computațional.

Dacă un algoritm este o propunere conceptuală pentru a rezolva o problemă, atunci programul reprezintă forma implementată într-un limbaj specific. De regulă un program rezolvă mai multe probleme, ceea ce duce la necesitatea divizării în subprograme numite module. Fiecare modul poate conține implementarea unui algoritm sau o îmbinare a mai multor algoritmi.

Orice algoritm trebuie să poată fi definit prin:

- Intare. Se poate ca un algoritm să nu conțină nicio dată ca intrare sau să aiba un set de date ce definește intrarea.

- Ieșire. Fiecare algoritm are definită cel puțin o ieșire ce reprezintă soluția oferită.
- Exprimare. Fiecare pas al algoritmului trebuie să fie clar.
- Finitidutine. Algoritmul trebuie să-și termine execuția după un număr finit de pași.
- Eficacitate. Fiecare pas trebuie să fie definit cât mai simplu pentru a putea fi executat rapid.

După modul de implementare, algoritmii se pot cataloga astfel:

- Recursiv-Iterativ. Recursivitatea înseamnă pe scurt apelul unei funcţii in interiorul ei. Pe de cealaltă parte, un algoritm iterativ, presupune execuţia succesivă a instrucţiunilor.
- Serial-Paralel. Modul în care se execută pașii definiți într-un algoritm poate fi consecutiv sau concomitent.
- Deterministic-Aleatoriu. Un algoritm deterministic va furniza pentru aceleași intrări un set de ieșiri care nu se va schimba oricâte rulări am avea. Pe de altă parte un algoritm aleatoriu, va produce pentru aceeași intrare la rulări diferite, ieșiri diferite.

În continuare vom prezenta un exemplu pentru a defini câteva modalități de a exprima un algoritm.

1.1 Exemplu introductiv

Pentru a exemplifica câteva moduri de a reprezenta pașii un algoritm putem folosi pentru început problema găsirii minimului într-un șir.

Iată cum putem defini problema găsirii minimului dintr-un șir:

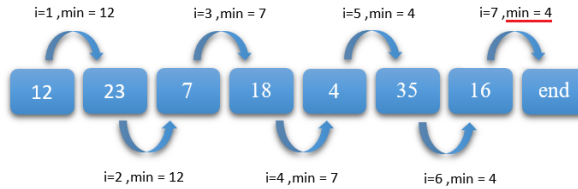
Intrare: O secvență de n numere **Ieșire:** Valoarea celui mai mic număr din secvență

Fie secvența $(12, 23, 7, 18, 4, 35, 16)$. Evident numărul minim se va afla parcurgând acest șir si reținând valoarea minimă, în acest caz 4. În figura 1.1 este reprezentată această parcurgere.

Există cel puțin trei tipuri de a exprima un algoritm astfel încât descrierea pașilor acestuia să nu depindă de niciun limbaj de programare:

1. Exprimare în limbaj natural. Aceasta presupune descrierea în cuvinte a pașilor.
2. Pseudocod. Descrierea are loc sub forma unui cod ce poate fi ușor transcris apoi în orice limbaj de programare.

(a) Sirul inițial. Cu gri sunt indicate indexurile valorilor din șir



(b) Parcurgerea șirului și reținerea minimului

Figura 1.1: Aflarea minimului dintr-un șir

3. Schemă logică. Un algoritm poate fi reprezentat sub o formă vizuală ușor de descris și de urmărit.

În cele ce urmează vom parcurge cele trei moduri de a implementa algoritmul de aflare a minimului dintr-un șir. Primul mod de a descrie un agloritm este tocmai expunerea pașilor în cuvinte precum în algoritmul 1.

Algoritm 1 Algoritm exprimat în limbaj natural

(I) Inițializează o variabilă *min* cu valoarea primului element din șir
(II) Parcurge tot restul șirului folosind o variabilă i pentru a incrementa poziția în șir
(III) Reține valoarea minimă în *min* comparând această variabilă cu fiecare element din șir

Al doilea mod este cel mai răspândit și anume pseudocodul din algoritmul 2. Vom detalia în secțiunea următoare regulile pentru a scrie un algoritm în pseudocod.

O schemă logică este o diagramă ce reprezintă grafic pașii unui algoritm folosind blocuri conectate de săgeți ce indică fluxul datelor. Blocurile sunt diferite forme geometrice ce semnifică instrucțiuni logice: dreptunghi-asignare, cerc-conector, romb-instrucțiune condițională, etc.

După cum se poate observa acest mod de descrie un algoritm este ceva mai complicat, se folosesc anumite notații specifice ceea ce implică respectarea unor reguli de scriere.

Algoritm 2 Algoritm exprimat în pseudocod

```
1:  procedure FIND_MIN(A)                          ▷ Find minimum in A
2:      min ← A[1]
3:      i ← 2
4:      n ← length(A)
5:      while i <= n do
6:          if min > A[i] then
7:              min ← A[i]
8:          end if
9:          i ← i + 1
10:     end while
11:     return min                                 ▷ The minimum of A is min
12: end procedure
```

Ultima modalitate de a descrie un algoritm este schema logică iar un exemplu este oferit mai jos în figura 1.2.

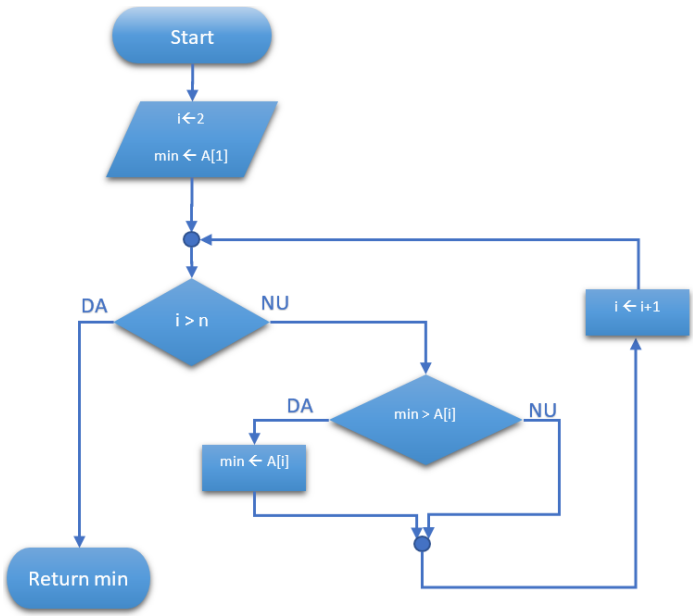


Figura 1.2: Aflarea minimului dintr-un șir. Schema logică.

1.2 Analiza algoritmilor

A analiza un algoritm poate însemna a prezice resursele necesare ca acesta să ruleze. Înainte de a analiza un algoritm, trebuie să avem un model al tehnologiei pe care va fi implementat. Vom presupune faptul că va rula pe un sistem cu un procesor, iar instrucţiunile vor fi executate una după alta, fără operaţii concurente. Înainte de a analiza algoritmii va trebui să stabilim regulile de exprimare a acestora.

1.2.1 Convențiile pentru pseudocod

Pentru a folosi corect și constant, în cele ce urmează, limbajul pseudocod, trebuie să ţinem cont de anumite reguli și anume:

1. Identarea indică o structură bloc (while, if, etc). În exemplul din algoritmul 2, corpul procedurii ce începe pe linia 2 constă din liniile 2-11, sau corpul while-ului ce începe pe linia 5 și conţine liniile 6-9, sunt exemple de instrucțiunii din cadrul unui bloc. În loc de identare se pot folosi paranteze, acolade sau specificatori begin, end pentru a spori claritatea codului.

2. Simbolul $\triangleright$ indică faptul că ceea ce urmează după el reprezintă un comentariu.

3. Atribuirea se face folosind operatorul „$\leftarrow$” .De exemplu i ia valoarea lui j se va scrie i $\leftarrow$ j. Ese posibil ca în alte notații atribuirea să se facă folosind „:=”.

4. Egalitatea se verifică folosind operatorul „=” .De exemplu i egal cu j se va scrie i = j

5. Variabilele (de exemplu i sau j) sunt locale în procedura respectivă. Nu vom folosi variabile globale în pseudocod.

6. Șirurile se vor reprezenta prin litere mari: A,B, accesarea unui element din șir se face utilizând paranteze pătrate ca de exemplu A[3]. Aici a fost accesat elementul cu indexul 3. Notaţia “..” este folosită pentru a indica un domeniu de valori din cadrul șirului. De exemplu A[1..j] indică subșirul ce este format din elementele A[1],A[2],...,A[j].

7. Parametrii sunt transmiși prin valoare, adică procedura primește copii ale parametrilor de la apel.

8. Anumite funcții predefinite pot fi referite direct prin numele lor fără a fi necesar descrierea formală a acestora. De exemplu $min(a, b, c)$ va returna valoarea cea mai mică dintre a,b și c.

9. Atributele unor obiecte vor fi indicate prin cuvinte în engleză cu font italic. De exemplu dacă tratăm un șir ca obiect, atunci atributul *length* va indica lungimea acestuia, adică numărul total de elemente ale șirului. Dacă dorim să reprezentăm un pointer, va trebui să marcăm acest lucru prin operaţia de atribuire. În alte notații lungimea șirului se indică folosind operatorul modul: $|A|$. Dacă avem două obiecte x și y, iar operaţia $y \leftarrow x$ produce egalitatea $f[x] = f[y]$ unde f este orice atribut al acestui obiect și apoi setăm $f[x] \leftarrow f[y]$ unde f este orice atribut al acestui obiect și setăm $f[x] \leftarrow 3$, aceasta va produce automat $f[y] = 3$.

10. Un obiect null va fi marcat cu valoarea NULL.

1.2.2 Analiza sortării prin inserție

Timpul necesar sortării unui șir folosind această metodă, depinde de intrare: sortarea unui șir de mii de elemente este evident, mai lentă, decât sortarea unui șir cu zece elemente. Pentru a putea cuantifica acest timp, în raport cu intrarea, va trebui să definim noţiunea de *timp de rulare* și noţiunea de *intrare* mai amănunţit.

Noţiunea de *mărime de intrare* depinde de problema studiată. De exemplu înmulţirea a doi întregi, este în strânsă relaţie cu numărul de biţi necesari reprezentării acelor numere. Pe de altă parte sortarea unui șir depinde de numărul de elemente ale acelui șir.

Noţiunea de *timp de rulare* înseamnă mai degrabă numărul de operaţii (pași) care trebuie executate pentru a rula algoritmul. Este mai bine să folosim termenul de *pas* pentru a nu lega noţiunea de operaţie de o arhitectură anume. Pentru a cuantifica cât mai corect timpul de rulare, vom introduce și noţiunea de constantă ce reprezintă timpul necesar execuţiei unei instrucţiuni.

De exemplu pentru a executa linia i, putem preciza constanta ca fiind timpul necesar execuţiei acelei linii. Vom denumi această constantă, costul unei operaţii. În continuare vom analiza, pornind de la algoritmul descris anterior timpul de execuţie al sortării prin inserţie:

Timpul de rulare al algoritmului este suma timpilor necesari rulării fiecărei operații. În cazul unei operaţii simple (ca în cazul for, se va înmulţi costul cu numărul de execuţii al acelei operaţii). Numărul de execuţii este ceva mai complicat în cazul operaţiei *while* din interiorul forului. Deoarece nu putem spune cu siguranţă de câte ori se va executa fiecare *while*, aceasta depinzând de **intrare** și anume de șirul de sortat. De aceea presupunem că t_j este numărul de execuţii în while pentru un anumit j.

Algoritm 3 Sortarea prin inserție

		const	timpi
1:	**procedure** INSERTION_SORT(A)		
2:	**for** j $\leftarrow$ 1 **to** n **do**	c_1	n
3:	$key \leftarrow A[j]$	c_2	$n-1$
4:	▷ insert $A[j]$ into sorted sequence $A[i..j-1]$	0	$n-1$
5:	$i \leftarrow j-1$	c_4	$n-1$
6:	**while** $i > 0$ AND $A[i] > key$ **do**	c_5	$\sum_{j=2}^{n} t_j$
7:	$A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
8:	$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
9:	**end while**		
10:	$A[i+1] \leftarrow key$	c_8	$n-1$
11:	**end for**		
12:	**end procedure**		

Putem încerca în continuare să stabilim numărul total de execuţii, și ca atare timpul de rulare total $T(n)$:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1) \tag{1.1}$$

În cel mai bun caz, și anume când șirul de intrare este deja sortat (instrucţiunea 5.) este executată doar pentru a efectua o verificare, și nu se vor executa niciodată instrucţiunile 6. și 7. În acest caz putem rescrie $T(n)$ considerând $\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} 1 = n-1$.

Asfel timpul total de rulare va fi:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1 + c_2 + c_4 + c_5 + c_8) - (c_2 + c_4 + c_5 + c_8) \tag{1.2}$$

Aceasta poate fi scrisă sub forma $an + b$ unde a și b sunt constante ce depind de c_i. Aceasta înseamnă că $T(n)$ este o funcție liniară de n.

În cazul în care șirul este scortat descrescător, calculând $T(n)$ obținem cel mai slab timp din punct de vedere al eficienței. Astfel instrucțiunea *while* va fi executată de fiecare dată iar $t_j = j$:

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \tag{1.3}$$

și

$$\sum_{j=2}^{n}(t_j - 1) = \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2} \tag{1.4}$$

Înlocuind în formula de calcul al lui $T(n)$ obținem:

$$\begin{aligned} T(n) = {} & c_1 n + c_2(n-1) + c_4(n-1) + \\ & c_5(\frac{n(n+1)}{2} - 1) + c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1) = \\ & (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned} \tag{1.5}$$

Acest timp poate fi exprimat sub forma $f = an^2 + bn + c$ cu a,b și c depinzând de constantele $c_{1..8}$.

1.2.3 Ordinul de timp

Am simplificat unele calcule pentru a ușura cât mai mult analiza algoritmului de inserţie. De exemplu, am ignorat costul concret al fiecărei instrucţiuni folosind constante de tip c_i. În cazul cel mai nefavorabil am ales constantele a,b și c pentru a arata faptul că polinomul este de ordinul 2. Pentru a simplifica și mai mult lucrurile vom proceda la următoarea notație. Se numește **ordin de creștere** acel timp de rulare dedus din funcția f ce reprezintă cel mai bine numărul de pași necesari execuției unui algoritm. Acest număr de pași este corelat cu timpul necesar rulării unui algoritm.

Din funcția f vom considera cel mai reprezentativ termen și anume an^2, deoarece pentru un n suficient de mare, ceilalți termeni vor fi mici relativ la an^2. Mai mult, vom ignora coeficientul, din moment ce aceștia sunt mai puţin semnificativi în determinarea ordinului de creștere. De aceea, obţinem pentru acest algoritm, în cel mai nefavorabil caz, timpul de rulare de $O(n^2)$, notaţie pe care o vom detalia în capitolul următor.

În cazul în care șirul de intrare ar fi fost deja sortat $T(n)$ ar fi fost sub forma $f = an + b$. Aceasta înseamnă că doar pentru acest caz timpul de rulare ar fi fost timpul de rulare de $O(n)$. Pentru a surprinde ambele cazuri există notații asimptotice pe care le vom studia ulterior.

De ce trebuie ca algoritmul să fie eficient, și ordinul de timp să fie cât mai mic? Pentru că un algoritm reprezintă o tehnologie. Astfel acesta va evolua inevitabil, în strânsă legătură cu platformele pe care a fost implementat. Independent însă de aceste platforme, un algoritm poate fi

îmbunătățit astfel încât să producă aceleași rezultate mai rapid ceea ce înseamnă un progres al tehnologiei folosite.

Luând în considerare o serie de factori precum limbajul în care este scris algoritmul, mediul în care va rula programul ce implementează algoritmul, putem extinde analiza de bază care va fi prezentată în acest curs, pentru a prezice cu acurateţe timpul necesar rulării unui program, pentru diferite intrări.

Totuși, în cadrul acestui curs vom ignora aceste aspecte, pentru a păstra simplitatea calculului, și a ușura înţelegerea modelelor descrise. În cele ce urmează, vom studia fundamentele matematice care ne vor fi de mare folos în analiza tuturor algoritmilor ce vor fi studiaţi.

1.3 Algoritmi fundamentali

În continuare, vom prezenta câţiva algoritmi nu foarte complicaţi, pentru a crea introducere pentru acest curs și pentru a exersa conceptele de mai sus și nu numai.

1.3.1 Înmulțirea a la russe

Este o operaţie de matematică cu origini în Egiptul antic, și este o metodă ce nu implică folosirea tabelului înmulţirii, ci doar divizarea cu 2 și adunarea. Marele avantaj al acestei metode este că poate fi implementată ușor în tehnica de calcul modernă, deoarece implică folosirea acestor două operaţii aritmetice de bază.

Iată un exemplu pentru a demonstra principiul, înmulţind numerele 52 și 15:

52	15	–
26	30	–
13	60	60
6	120	–
3	240	240
1	480	480
		780

Tabela 1.1: Înmulțirea a la russe pentru numerele 52 și 15

După cum se poate observa în tabelul 1.1, se aplică în mod repetat împarțirea cu 2 a primului număr și inmulțirea cu 2 a celui de al doilea număr.

În cazul în care împărțirea cu 2 a primului număr a avut ca rezultat un număr impar (în prima coloană), se reține rezultatul înmulțirii cu 2 a celui de-al doilea număr (în a doua coloană). În a treia coloană stocăm aceste rezultate parțiale. La final însumăm numerele din a treia coloană și acesta este produsul dintre 52 și 15.

În cele ce urmează vom descrie acest algoritm (4) folosind pseudocod.

Algoritm 4 Înmulțirea a la russe

```
1:  procedure RUSSE(a, b)
2:      arrays X, Y
3:      X[1] ← a; Y[1] ← b
4:      i ← 1
5:      ▷ se construiesc cele două coloane
6:      while X[i] > 1 do
7:          X[i + 1] ← X[i]/2
8:          Y[i + 1] ← Y[i] + Y[i]
9:          i ← i + 1
10:     end while
11:     prod ← 0
12:     while i > 0 do
13:         if X[i]%2 = 0 then
14:             prod ← prod + Y[i]
15:         end if
16:         i ← i − 1
17:     end while
18: return prod
19: end procedure
```

Ordinul de timp al acestui algoritm este *O(n)*, iar pseudocodul prezentat poate fi optimizat adică se poate elimina a doua buclă **while** și de asemenea se pot elimina cele două șiruri auxiliare X,Y.

Întrebarea firească este de ce funcționează acest algoritm? Răspunsul poate fi dat transformând numerele în sistemul binar. În această bază, împărțirea la 2 este echivalentul șiftării tuturor biților la dreapta cu o unitate. Asemenea înmulțirea cu 2 reprezintă șiftarea tuturor biților la stânga cu o unitate și completarea cu 0 a bitului cel mai din dreapta.

De exemplu numărul 14 în binar este 1110. Dacă ar fi să îl înmulțim cu 1 tot în binar, acesta devine: $14 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$. Se poate observa că doar puterile lui 2 înmulțite cu 1 contează în suma finală.

Mai departe să înmulțim 14 cu 2. Adică 1110 cu $\mathbf{1}0$ în binar: $14 * 2 = 28 + 0 = 11100 * \mathbf{1} + 1110 * 0 = 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$. După cum se poate vedea, acel *0* marcat cu font italic nici nu a contat în suma

finală. De aceea nici algoritmul nu ia in calcul numere pare din coloana a doua.

Această metodă are aplicații în domeniul educațional, fiind o metodă ușor de înțeles și de implementat, dar și în domeniul ingineresc deoarece aduce o îmbunătărire a timpului de execuție acolo unde numerele sunt foarte mari. Această problemă poate fi încadrată la categoria *divide et impera* despre care vom vorbi în capitolul dedicat acestui tip de rezolvare.

1.3.2 Sortarea prin selecție

Algoritmul are ca intrare un șir de date, nu neapărat sortat, și se cere sortarea crescător a acestuia. Spre deosebire de sortarea prin inserţie, sortarea prin selecţie, lucrează altfel, plasând la fiecare pas un element pe poziţia lui finală.

Algoritm 5 Sortare prin selecție

```
1:  procedure SELECTION_SORT(A[1..n])
2:      for i ← 1 to n-1 do
3:          minj ← i; minx ← A[i]
4:          ▷ caut poziția finală a lui A[i] în șir
5:          for j ← i+1 to n do
6:              if A[j] < minx then
7:                  minj ← j
8:                  minx ← A[j]
9:              end if
10:         end for
11:         ▷ la final schimb elementul actual cu cel mai mic găsit
12:         A[minj] ← A[i]
13:         A[i] ← minx
14:     end for
15: end procedure
```

Algoritmul 5 funcţionează astfel: pornind de la primul element în șir, ne folosim de variabile auxiliare *minj* și *minx* pentru a reţine poziţia celui mai mic element respectiv valoarea acestuia. Apoi vom parcurge restul șirului căutând cel mai mic element din subșirul care este format din elementele ce urmează elementului actual. Găsim cea mai mică valoare, reţinem poziţia, și facem interschimbarea cu elementul actual (daca este cazul). Continuăm până când am parcurs șirul până la penultimul element, clipă în care șirul va fi deja sortat.

Pentru exemplificare vom utiliza ca intrare șirul {5, 2, 4, 6, 1, 3 }, pașii acestui algoritm fiind descriși, vizual, în figura 1.3.

Ordinul de timp al acestui algoritm este $O(n^2)$, independent de ordonarea iniţială a elementelor.

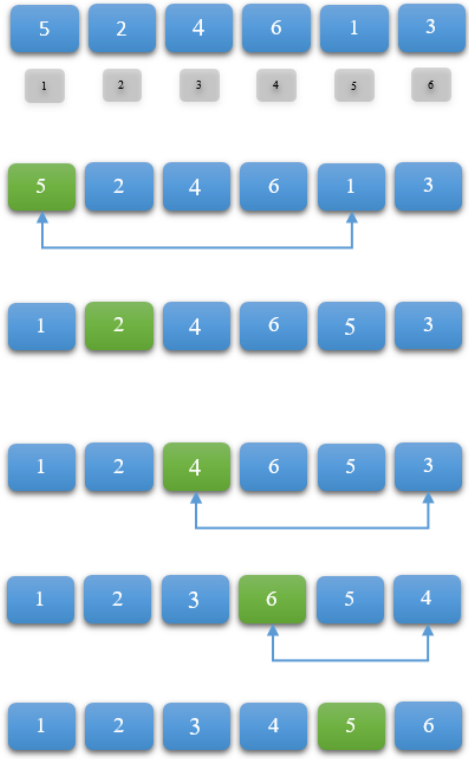


Figura 1.3: Exemplu pentru algoritmul de sortare prin selecție

Acest tip de sortare este destul de rapid pentru șiruri relativ mici (sub 100 elemente). Sortarea fișierelor sau a folderelor, verificarea unicității unor elemente, selecția rapidă a celui de-al k element dintr-un șir ordonat sunt câteva exemple de aplicații ale sortării. Vom vedea în capitolele ce urmează, diferite optimizări ale acestei probleme.

1.3.3 Șirul lui Fibonacci

Șirul lui Fibonacci este definit prin următoarea recurenţă:

$$\begin{cases} f_0 = 0, f_1 = 1 \\ n < 2 f_n = f_{n-1} + f_{n-2} \quad n \geq 2 \end{cases} \tag{1.6}$$

Acest șir a fost descoperit de Leonardo Pisano, cunoscut sub numele Leonardo Fibonacci. Cel de-al n-lea termen din șir se poate scrie folosind definiţia și anume conform algoritmului 6.

Algoritm 6 Calculul termenului n al șirului Fibonacci.
Varianta recursivă

```
1: procedure FIBO_R(n)
2:     if n < 2 then   return n
3:     else  return FIBO_R(n-1) + FIBO_R(n-2)
4:     end if
5: end procedure
```

Metoda este extrem de ineficientă, deoarece recalculează de mai multe ori aceleași valori, ordinul de timp fiind unul exponenţial $O(\phi^n)$. Vom lămuri în capitolul dedicat calculului complexității algoritmilor. Acest algoritm introduce noţiunea de recursivitate, asupra căreia vom insista ulterior în curs. Motivul pentru care acest algoritm este extrem de ineficient este pentru că în calculul termenului n se repetă (inutil) calculul termenilor inferiori după cum reiese din figura 1.4.

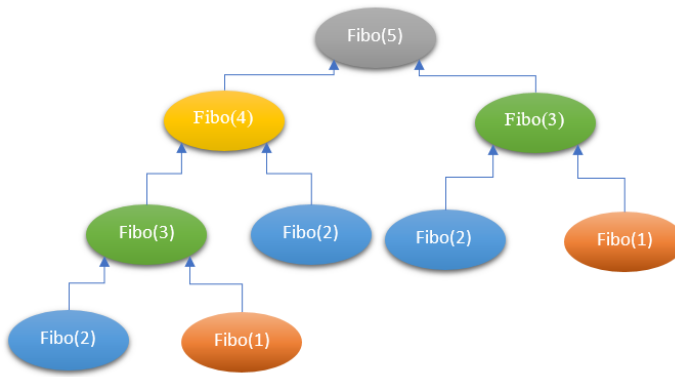


Figura 1.4: Calculul termenului 5 al șirului Fibonacci folosind algoritmul 6.

O altă metoda mai eficientă de a rezolva această problemă, și anume într-un timp liniar $O(n)$, este prezentată în algoritmul 7. Această versiune reprezintă defapt o metodologie de a elabora algoritmi și anume *programarea dinamică*, despre care vom vorbi în capitolul rezervat acesteia.

Algoritm 7 Calculul termenului n al șirului Fibonacci.
Varianta iterativă

```
1: procedure FIBO_I(n)
2:     i ← 0; j ← 1
3:     s ← 1
4:     for k ← 1 to n do
5:         i ← j
6:         j ← s
7:         s ← i + j
8:     end for
9: return i
10: end procedure
```

Secvența Fibonacci are o semnificație mai mare decât cea educațională. În lumea naturală există diferite specii de plante care se dezvoltă în conformitate cu această serie sau au în componență părți ce pot fi aproximate cu ajutorul acestui șir. De asemenea, anumite metodologii de dezvoltare și estimare de applicații software, folosesc seria Fibonacci pentru a determina timpul necesar rezolvării unui proiect.

1.3.4 Algoritmul lui Euclid

Algoritmul ce poartă numele lui Euclid (posibil să fi fost descoperit înaintea matematicianului grec) este o metodă eficientă de a calcula CMMDC (cel mai mare divizor comun) a două numere întregi.

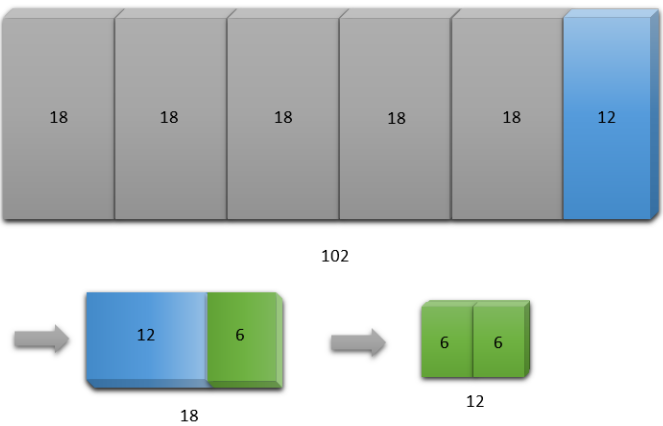


Figura 1.5: Găsirea CMMDC-ului dintre numerele 102 și 18. Reprezentare vizuală

Algoritmul 8 se bazează pe principiul că divizorul a două numere nu se schimbă dacă extragem numărul cel mai mic din cel mai mare. De exemplu CMMDC-ul lui 102 si 18 este 6. Dacă extragem 18 din 102 obţinem 84, iar CMMDC-ul lui 18 și 84 este tot 6. În figura 1.5 se poate observa calculul pentru a găsi CMMDC-ul dintre numerele alese pentru exemplificare și anume 102 și 18.

Algoritmul lui Euclid se bazează pe această proprietate și spune că CMMDC-ul a două numere se poate afla astfel: se află restul împărţirii celui mai mare număr la cel mai mic și se reţine acest rest, câtul și vechiul deîmpărţit devin noile numere cărora le aplicăm aceeași operaţie, până când restul devine zero. Penultimul rest este cmmdc-ul numerelor iniţiale.

Algoritm 8 Algoritmul lui Euclid

```
1: procedure EUCLID(m, n)
2:     while n ≠ 0 do
3:         temp ← n
4:         n ← m%n
5:         m ← temp
6:     end while
7: return m
8: end procedure
```

Pentru exemplul ales mai sus și anume 102 și 18 algoritmul va funcționa astfel:

```
temp ← 18
n ← 12
m ← 18

    temp ← 12
    n ← 6
    m ← 12

        temp ← 6
        n ← 0
        m ← 6
```

Vom vedea că acest algoritm poate fi aplicat pentru a rezolva o problemă de tip *bin packing* folosind metodologia *greedy*.

Aplicațiile practice ale acestui algoritm variază de la simplificarea reprezentării fracțiilor, la construcția sistemului de criptare RSA [4] sau la elaborarea strategiilor de tip *look-ahead* precum [1].

1.3.5 Turnurile din Hanoi

Problema turnurilor din Hanoi reprezintă un puzzle matematic format din 3 tije și n discuri situate inițial pe prima tijă. Aceste discuri sunt plasate astfel încât la bază se află discul cu cel mai mare diametru, deasupra lui discul cu următorul diametru ca mărime ș.a.m.d.

Scopul este de a transfera discurile pe ultima tijă (a treia) astfel încât niciodată să nu avem situația în care, pe orice tijă ne-am afla, să existe un disc cu diametru mai mare deasupra unui disc cu diametru mai mic.

Altfel spus, fie n discuri 1,2,3,... cu diametrele d_1,d_2,d_3,... situate pe prima tijă, astfel încât $d_1 > d_2 > d_3$..., să se plaseze pe ultima tijă n discuri 1,2,3... astfel ca ordinea să se mențină $d_1 > d_2 > d_3$... pe a treia tijă.

Problema își are originile într-un templu indian din Varanasi, oraș cunoscut anterior ca Benares. Preoții aveau această sarcină de a muta $n = 64$ de discuri de aur dintr-o parte a templului în alta, folosind o locație intermediară pentru a plasa aceste discuri. Deoarece discurile erau fragile, trebuia să se mențină ordinea indicată mai sus.

Vom vedea ulterior că algoritmul are ordinul de timp $O(n) = 2^n$ ceea ce înseamnă că ar fi avut de efectuat 2^64 pași. De aceea se spune că atunci când, ipotetic, ar fi terminat această sarcină, ar fi venit de mult sfârșitul lumii. Ceea ce nu este departe de adevăr pentru că această sarcină ar necesita un efort de sute de miliarde de ani, în cazul în care este executată de o singură persoană.

Iată câteva motive pentru care această problemă este relevantă:

- este un exemplu foarte bun de recursivitate
- jocul este folosit de neurofiziologi pentru a verifica defectele din lobul frontal
- ajută la stocarea datelor pentru a minimiza timpul de re-folosire al acestora
- este întâlnită și în lumea animală: o specie de furnici a folosit acest algoritm, într-un experiment efectuat în 2010 în care se observa căutarea celui mai scurt drum [3]

În figura 1.6 am reprezentat o soluție vizuală pentru acest algoritm cu $n = 3$ discuri.

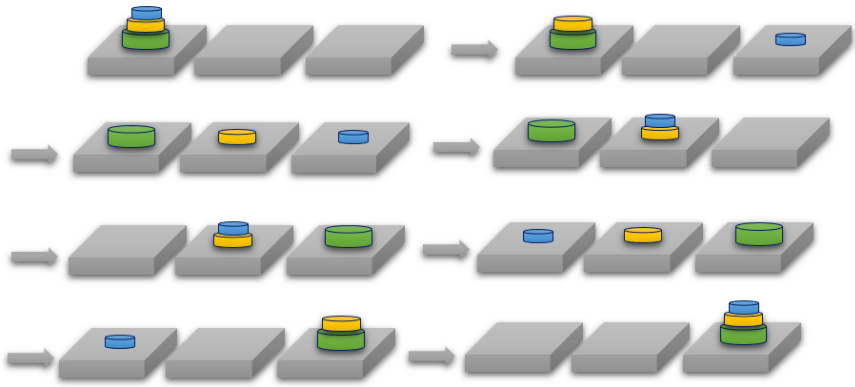


Figura 1.6: Rezolvarea algoritmului turnurilor din Hanoi. Reprezentare vizuală.

Iată și rezolvarea formală a acestei probleme:

- Se etichetează tijele cu A, B și C
- se numerotează discurile 1 - cel mai mic, n- cel mai mare

Algoritmul recursiv este 9:

Algoritm 9 Algoritmul turnurilor din Handoi

```
1: procedure HANOI(n, A, B, C)
2:     if n ≠ 0 then
3:         HANOI(n − 1, A, B, C)
4:         afisez «Mută discul de pe» A «pe» C
5:         HANOI(n − 1, B, C, A)
6:     end if
7: end procedure
```

Observăm că problema poate fi descompusă în trei subprobleme independente și anume mutarea a $n-1$ discuri, mutarea de pe A pe C și apoi mutarea a $n-1$ discuri. Din acest punct de vedere problema poate fi încadrată ca fiind *divide et impera*.

2 Structuri de date de bază

Pe parcursul acestui curs vom vorbi despre algoritmi a căror implementare necesită utilizarea unor structuri de date specifice. Ca atare acest capitol va trata structuri precum liste, tabele hash, grafuri, arbori ș.a.m.d.. Putem privi structurile de date ca un pas premergător studiului algoritmului deoarece există o interdependență între algoritm și structura aleasă.

2.1 Șiruri

Un șir A este o structură lineară alocată ca un bloc omogen de date în care datele sunt poziționate în locații consecutive. Indexarea acestor locații începe cu poziția 1 și se termină cu $n = length(A)$.

Aceste date pot avea un tip standard, precum întreg, string, caracter etc. Figura 1.1 prezintă cel mai bine un șir de 7 întregi cu valorile 12, 23, 7, 18, 4, 35, 16. De regulă, nu există nici o restricție cu privire la relațiile între date (exemplu: ordonare, unicitate) într-un șir, cu excepția în care problema dată specifică aceste restricții.

Avantajul folosirii șirurilor este acela că memoria este alocată o singură dată. Totodată accesul la un indice i din șir se poate realiza în $O(1)$ în cazul folosirii pointerilor.

Dezavantajul constă în proprietatea tablourilor de a fi imuabile: odată ce memoria este alocată nu se mai poate schimba numărul de elemente din șir.

În continuare vom studia câteva dintre operațiile cu șiruri.

2.1.1 Căutarea în șir

Căutarea unui element într-un șir este relativ simplu și presupune o parcurgere. Procedura se numește căutare lineară datorită ordinului de timp $O(n)$. Algoritmul pentru aceasta este 10.

Algoritm 10 Algoritmul de căutare în șir

```
1: procedure CĂUTARE_LINEARĂ(A[1..n], x)
2:     flag ← false
3:     i ← 1
4:     while i ≤ n do
5:         if A[i] = x then
6:             flag ← true
7:             break
8:         end if
9:         i ← i + 1
10:    end while
11:    if flag = false then
12:        print «Not found»
13:    end if
14: end procedure
```

2.1.2 Inversarea elementelor într-un șir

Algoritmul ce inversează ordinea inițială a elementelor este 11.

Algoritm 11 Algoritmul de inversare a elementelor într-un șir

```
1: procedure INVERSARE(A[1..n])
2:     i ← 1
3:     while i ≤ n/2 do
4:         aux ← a[i]
5:         a[i] ← a[n − i − 1]
6:         a[n − i − 1] ← aux
7:         i ← i + 1
8:     end while
9: end procedure
```

Figura 2.1 prezintă acest algoritm pornind de la șirul ales la începutul secțiunii.

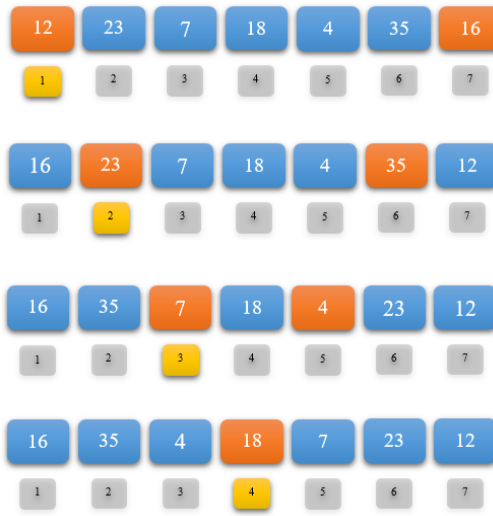


Figura 2.1: Inversarea elementelor într-un șir.

Cu galben s-a reprezentat poziția indexului i pe măsură ce acesta avansează către mijlocul șirului.

Acest algoritm este de tipul "in-place" deoarece pașii acestuia nu necesită utilizarea unei alte structuri, valorile fiind înlocuite în cadrul aceluași șir inițial. Vom vedea că acest exercițiu este util mai ales în cazul unor sortări.

2.1.3 Șiruri bidimensionale

Un șir 2D este o matrice. Pentru a parcurge și a accesa elementele unei matrice avem nevoie de 2 indecși după cum este indicat în algoritmul 12.

Algoritm 12 Parcurgere și inițializare elemente matrice

```
1: procedure MATRIX_INIT(A, n, m)
2:     for i ← 1 to n do
3:         for j ← 1 to m do
4:             A[i][j] ← i * j
5:         end for
6:     end for
7: end procedure
```

O matrice A cu n linii și m coloane poate fi parcursă într-un timp $O(mn)$ deoarece avem un for cu m în cadrul unui alt for cu n instrucțiuni.

În figura 2.2 avem un reprezentarea vizuală a algoritmului 12 pentru o matrice de 3×4.

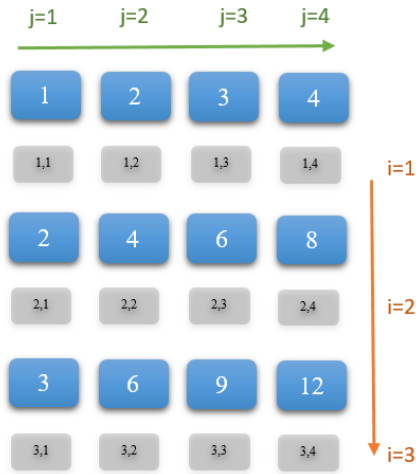


Figura 2.2: Parcurgerea și inițializarea elementelor unei matrice.

2.1.4 Matrice rare

O matrice rară este un tip special de matrice în care majoritatea elementelor sunt zero. Stocarea fiecărei valori într-o structură de tip matrice devine astfel ineficientă. Astfel o optimizare constă în memorarea valorilor non-zero și a pozițiilor acestora.

Pentru matricea din figura 2.3 putem creea o structură de tip matrice cu 3 coloane și atâtea linii câte elemente non-zero sunt în matricea inițială. Tabelul 2.1 conține soluția de reprezentare a matricii sparse.

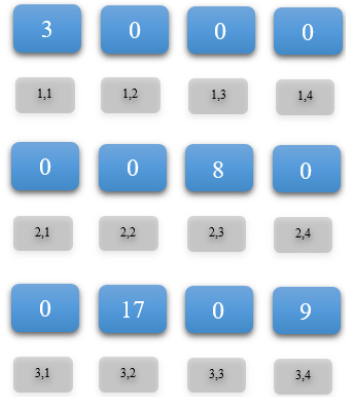


Figura 2.3: Matricea rară inițială.

3	1	1
8	2	3
17	3	2
9	3	4

Tabela 2.1: Reprezentarea unei matrici rare

Metoda este eficientă deoarece în loc de 12 locații, necesită doar 4, parcurgerea făcându-se de trei ori mai rapid în acest caz. Totdată, matricea originală poate fi refăcută oricând, dacă este necesar.

2.2 Liste

O listă înlănțuită este o structură ce are ca fundație *nodul*. Un nod este o entitate formată din două părți: legătura și data. În dată sunt stocate valorile nodurilor care pot fi de simple: intregi, float-uri, caractere sau complexe precum obiecte de diverse tipuri.

O reprezentare în pseudocod a unui nod cea de mai jos:

```
struct {
Data d;
Node next;
} Node;
```

Aici d are tipul *Data* adică poate fi orice tip de dată, așa cum s-a indicat mai sus, iar *next* este legătura către următorul element din listă. În cazul în care în listă nu există decât un element, valoarea lui *next* este $NULL$.

O reprezentare vizuală a unei liste înlănțuite este în figura 2.4 unde avem 3 elemente. Primul element se mai numește *capul* listei iar ultimul element conține în *next* valoarea $NULL$.

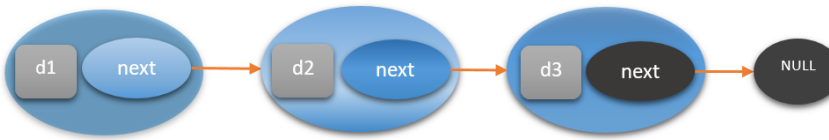


Figura 2.4: Lista simplu înlănțuită.

Lista din figura de 2.4 este o listă simplu înlănțuită adică fiecare element are legătură către un singur element din listă și anume către următorul. Această listă notată cu L conține 3 elemente dintre care cel cu data $d1$ este primul element din listă, cel cu data $d3$ fiind evident ultimul element din listă. Vom vedea în continuare că listele fi dublu și multiplu înlănțuite și cum anume se realizează aceasta.

Orice listă trebuie să permită implementarea celor 4 operațiuni de bază:

- Inserare
- Modificare
- Căutare/Parcurgere

- Ștergere

În cele urmează vom trata fiecare operațiune în parte.

2.2.1 Operațiuni cu liste simplu înlanțuite

Inserarea

Pentru a crea un element și a-l adăuga într-o listă, trebuie mai întâi să alocăm memorie pentru noul nod și apoi să stabilim locația din listă, unde va fi acesta inserat. Pentru a insera un element la începutul listei va trebui ca el să devină primul element și legătura sa *next* să fie către restul listei.

Algoritmul pentru inserare la începutul listei este 13.

Algoritm 13 Inserare la începutul listei

```
1:  procedure INSERT_BEGINNING(L, val)
2:          ▷ Alocăm memorie pentru noul nod
3:      newnode ← new Node
4:          ▷ Atribuim valoarea specifică acestuia
5:      newnode.d ← val
6:      ▷ Nodul are ca referință toată lista inițială
7:      newnode.next ← L
8:              ▷ El devine lista.
9:      L ← newnode
10: end procedure
```

Figura 2.5 prezintă inserarea unui nou nod în lista L. Noul nod (cu valoarea d) va deveni capul listei L. Complexitatea acestui algoritm este $O(1)$ deoarece nu necesită decât 4 pași.

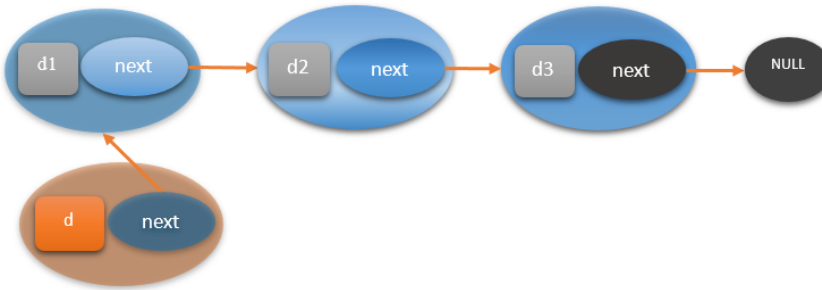


Figura 2.5: Inserarea unui nou element la începutul listei.

Pentru a insera un element la sfârșitul listei va trebui mai întâi să parcurgem lista și apoi să fie referențiat de ultimul nod din listă. Astfel noul nod devine ultimul element din listă.

Algoritmul pentru inserarea unui element în listă, la finalul acesteia, este 2.6.

Algoritm 14 Inserare la finalul listei

```
1:  procedure INSERT_END(L, val)
2:          ▷ Alocăm memorie pentru noul nod
3:      newnode ← new Node
4:          ▷ Atribuim valoarea specifică acestuia
5:      newnode.d ← val
6:          ▷ Parcurgem lista L folosind un nod auxiliar
7:      auxnode ← L
8:      while auxnode.next ≠ NULL do
9:          auxnode ← auxnode.next
10:     end while
11:         ▷ La final inserăm nodul în listă
12:     auxnode.next ← newnode
13: end procedure
```

Inserarea la finalul listei are reprezentarea vizuală în figura 2.6. Noul element se inserează prin rescrierea *next*-ului ultimului element. Noul element indică automat către *NULL* încă din momentul creării lui.

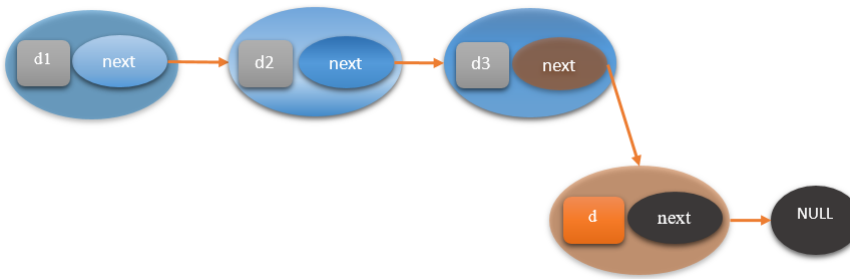


Figura 2.6: Inserarea unui nou element la finalul listei.

Ordinul de timp al acestui algoritm este $O(n)$ deoarece trebuie parcursă întreaga listă.

Pentru a insera un nod după orice poziție din listă va trebui să parcurgem lista până la elementul căutat și va trebui să-i refacem referințele

acestuia. Totodată noul nod va trebui să pointeze către următorul element, cel de după poziția căutată.

Algoritm 15 Inserare în interiorul listei

```
1:  procedure INSERT_END(L, x, val)
2:          ▷ Alocăm memorie pentru noul nod
3:      newnode ← new Node
4:          ▷ Atribuim valoarea specifică acestuia
5:      newnode.d ← val
6:          ▷ Parcurgem lista L folosind un nod auxiliar
7:      auxnode ← L
8:      ▷ până când găsim elementul după care dorim inserția
9:      while auxnode.d ≠ x do
10:         auxnode ← auxnode.next
11:     end while
12:   ▷ Inserăm nodul în listă suprascriind 2 referințe:
13:         ▷ noul nod pointează către restul neparcurs al listei
14:     newnode.next ← auxnode.next
15:         ▷ și nodul anterior va pointa către noul nod
16:     auxnode.next ← newnode
17: end procedure
```

Inserarea în interiorul listei are reprezentarea vizuală în figura 2.7. Noul element se inserează prin rescrierea celor 2 legături marcate cu verde. Se presupune că $x = d2$.

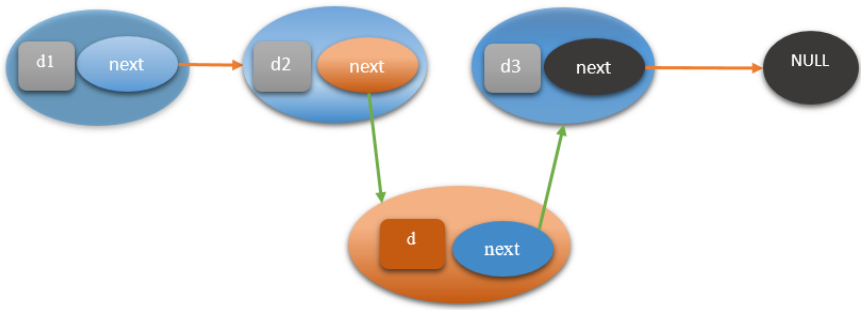


Figura 2.7: Inserarea unui nou element la finalul listei.

Modificarea unui element din listă

Modificarea valorii unui element din listă poate fi realizată prin cel puțin două strategii. Dacă se cunoaște valoarea elementului din listă se parcurge lista exact ca în algoritmul 15. Atunci când elementul este găsit, se modifică valoarea d a obiectului.

Dacă se cunoaște poziția în listă a elementului după care se dorește inserarea, atunci algoritmul 16 este mai indicat.

Algoritm 16 Modificarea unui elementdin listă

```
1:  procedure ChangeAtPosition(L, poz, val)
2:      ▷ Parcurgem lista L folosind un nod auxiliar
3:      auxnode ← L
4:      i ← 1
5:      ▷ până când găsim elementul de modificat
6:      while poz ≠ i do
7:          i ← i + 1
8:      end while
9:      ▷ Modificăm valoarea nodului curent
10:     auxnode.d ← aval
11: end procedure
```

Evident algoritmul pornește de la prezumția că poziția introdusă poz este mai mică decât numărul de elemente din listă. Se poate adăuga o validare la începutul algoritmului care să verifice acest lucru, în caz contrar restul procedurii să nu se execute.

Parcurgerea unei liste

Algoritmul de parcurgere a unei liste este foarte asemănător celor de mai sus și anume reprezentat de 17.

Algoritm 17 Parcurgerea unei liste

```
1: procedure Traverse(L, poz, val)
2:     ▷ Parcurgem lista L folosind un nod auxiliar
3:     auxnode ← L
4:     while auxnode ≠ NULL do
5:         Print auxnode.d
6:         auxnode ← auxnode.next
7:     end while
8: end procedure
```

Scopul acestei parcurgeri este de a afișa toate elementele unei liste. În cazul în care se dorește căutarea unui element de o anumită valoare atunci se va modifica linia 5.

Ștergerea unui element din listă

Ca și în cazul inserării, există 3 posibilități: ștergerea primului element, la final și în interiorul listei.

Ștergerea primului element din listă înseamnă defapt înlocuirea listei cu elementul următor.

Algoritm 18 Ștergere la începutul listei

```
1: procedure DELETE_BEGINNING(L)
2:         ▷ Preluăm lista L folosind un nod auxiliar
3:     auxnode ← L
4:         ▷ Ștergem primul nod, ignorând-ul
5:     L ← auxnode.next
6: end procedure
```

Conform figurii 2.8, noul *cap* al listei devine elementul cu data *d*2. Legătura care se distruge prin instrucțiunea 4 din algoritmul 18 este cea indicată cu roșu în figură.

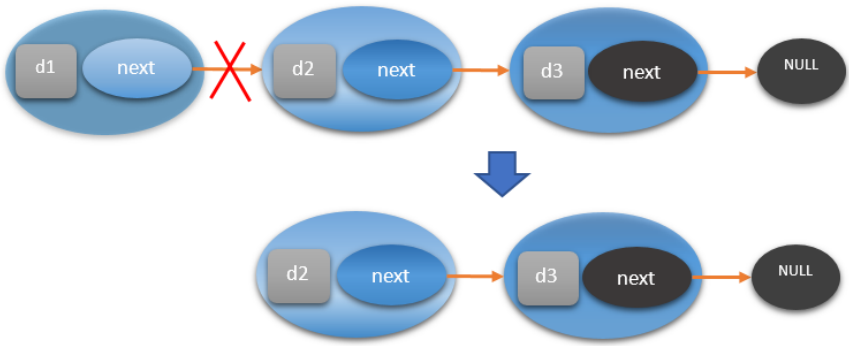


Figura 2.8: Ștergerea unui element de la începutul listei.

Ștergerea ultimului element din listă implică parcurgerea listei până la ultimul element care are *next*-ul nenull. Ștergerea se realizează prin ignorarea referinței acestui element.

Algoritm 19 Ștergere la finalul listei

```
1: procedure DELETE_END(L)
2:       ▷ Parcurgem lista L folosind un nod auxiliar
3:     auxnode ← L
4:     while auxnode.next ≠ NULL do
5:         auxnode ← auxnode.next
6:     end while
7:     auxnode ← NULL
8: end procedure
```

Figura 2.9 prezintă ștergerea de la sfârșitul listei. Lista se va parcurge de la început până la elementul $d3$ iar apoi acesta devine $NULL$. Automat după această instrucțiune, elementul $d2$ devine ultimul element din listă.

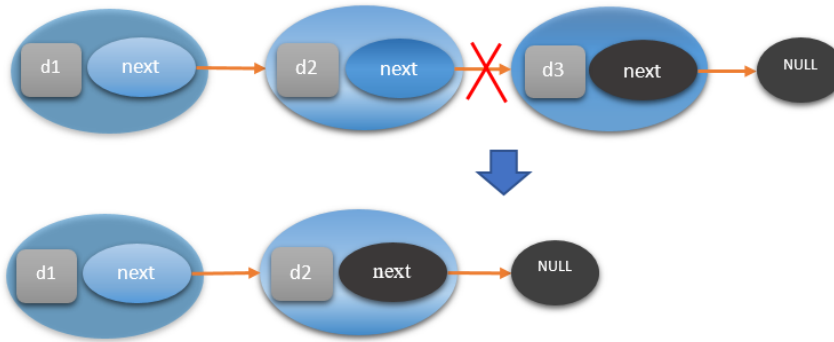


Figura 2.9: Ștergerea unui element de la finalul listei.

Ștergerea unui element din interiorul listei are loc astfel: se parcurge lista până la elementul de șters și apoi se elimină acesta prin "saltul" peste el. Adică legătura elementului anterior va fi către elementul posterior nodului de șters.

Algoritm 20 Ștergere în interiorul listei

```
1: procedure DELETE_END(L, val)
2:       ▷ Parcurgem lista L folosind un nod auxiliar
3:     auxnode ← L
4:     ▷ până când data următorului element va fi egală cu val
5:     while auxnode.next.d ≠ val do
6:         auxnode ← auxnode.next
7:     end while
8:   ▷ Atunci când am găsit elementul de șters, "sărim peste el"
9:     auxnode.next ← auxnode.next.next
10: end procedure
```

Figura 2.10 prezintă ștergerea elementului $d2$ prin refacerea legăturii de la $d1$ către $d3$, legătură marcată cu verde.

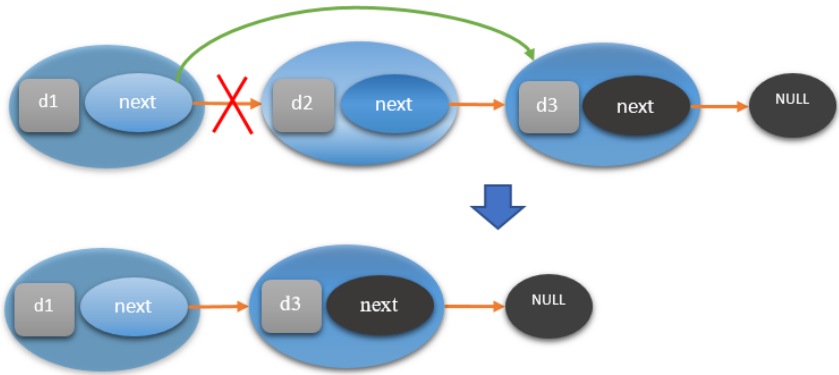


Figura 2.10: Ștergerea unui element din interiorul listei.

Avantajele listelor sunt următoarele:

- Sunt dinamice: se pot extinde, micșora cu ușurință, spre deosebire de șiruri care sunt imutabile
- Sunt ușor de implementat, deoarece nu au restricții
- Sunt structuri de bază pentru alte tipuri de date precum stiva sau coada

Dezavantajul principal intervine atunci când se impun anumite restricții asupra datelor elementelor din listă, ceea ce face ca această structură să nu fie cea mai eficientă.

Spre exemplu în mulțimi elementele duplicat nu au voie să existe, ceea ce înseamnă că la inserare va trebui să parcurgem întreaga listă indiferent de locația la care se inserează. Dacă lista va conține elemente ordonate, va trebui de asemenea parcursă lista pentru a insera la locația corectă.

De aceea, pentru astfel de operațiuni există alte structuri mai eficiente precum *tabele hash* sau *arbori balansați*.

În cele ce urmează vom detalia două dintre cele mai populare structuri derivate din liste și anume stiva și coada.

Stiva și coada sunt mulţimi dinamice în care un element va fi adăugat sau șters prin intermediul unor operaţii specifice. Într-o stivă, elementul șters va fi acela care a fost introdus cel mai recent în structură. Astfel stiva este o structură de dată de tip last-in first out, sau LIFO. Pe de altă parte, coada este structura de date în care ștergerea are loc conform regulii: first-in, first-out sau FIFO.

2.2.2 Stiva

Stiva este o structură de tip LIFO ce poate fi accesată și modificată prin două operațiuni *PUSH* și *POP*. Această structură poate fi implementată ca un șir sau ca o listă înlănțuită.

Vom trata cazul al doilea deoarece este cel mai dinamic mod de a lucra cu stive.

Operațiunea *PUSH* este defapt operțiunea de inserare a unui element nou la începutul listei, reprezentat de algoritmul 13 și figura 2.5.

Operațiunea *POP* este defapt operțiunea de ștergere a unui element de la începutul listei, reprezentat de algoritmul 18 și figura 2.8.

Aplicațiile acestei structuri sunt deosbit de numeroase precum:

1. inversarea facilă a șirurilor
2. implementarea mecanismelor *undo* din aplicații
3. oferă posibilitatea revenirii într-un punct atunci când se iau anumite decizii
4. compilatoarele folosesc evaluarea expresiilor folosind stiva
5. o serie de limbaje de programare folosesc stive pentru a implementa apelul de subrutine
6. sunt folosite în cadrul multor altor agloritmi

Evident cele două operații de *PUSH* și *POP* pot ridica excepții. *Underflow* este excepția ce se obține atunci când încercăm să scoatem elemente dintr-o stivă goală. *Overflow* este excepția ce se obține atunci când încercăm să adăugăm elemente într-o stivă plină. Faptul că o stivă este plină poate fi stabilit printr-o limită: fie numărul maxim de elemente, fie memoria maximă alocată tuturor elementelor din stivă.

În cele ce urmează vom exemplifica folosirea stivei ca structură de dată pentru rezolvarea formei postfixate a unei expresii aritmetice.

Expresii aritmetice postfixate

Notaţia infixată a unei expresii este 3+4 iar notaţia prefixată sau denumită și poloneză este + 3 4. Denumirea de forma poloneză provine de la matematicianul polonez Jan Lukasiewicz.

De ce folosim această notaţie? În forma poloneză, operatorii sunt plasaţi după operanzi, după cum am arătat mai sus. Dacă sunt operaţii multiple, operatorul va urma după al doilea operand, așa încât expresia $3 - 4 +$

5 va fi scrisă 3 4 − 5 +marcând faptul că scădem 4 din 3 și apoi adunăm la suma nou obţinută 5. Avantajul acestei notaţii este că elimină necesitatea parantezelor din notaţia infixată.

De exemplu expresia 3−4∗ 5 poate fi scrisă ca 3 − (4 ∗ 5), dar scrisă ca (3 − 4) ∗ 5 are cu totul alt înţeles și rezultat. În forma posfixată vom scrie expresia 3 − 4 ∗ 5 ca 3 4 5 ∗ − ce înseamnă clar, 3 (4 5 ∗) − adică 3 20 −. Cum funcţionează? Pentru evaluarea unei expresii operanzii sunt plasaţi pe o stivă, iar în momentul efectuării operaţiei, operanzii sunt scoși, cu ei se va efectua operaţia, iar rezultatul acesteia va fi plasat înapoi în stivă. Practic la final, valoarea expresiei postfixate, se află în vârful stivei.

Aplicații practice ale formei postfixate

- Calculele au loc imediat ce operatorul este specificat. În acest fel, expresiile nu sunt evaluate ca un bloc de la stânga la dreapta, ci calculate bucată cu bucată, eficientizând timpul de calcul.
- Stiva permite stocarea unui rezultat intermediar pentru a fi folosit mai târziu, ceea ce permite calculatoarelor ce folosesc această formă, să evalueze expresii de orice complexitate, spre deosebire de calculatoarele algebrice.
- Parantezele de orice tip nu mai sunt necesare, calculele sunt deci mai simplu efectuate.
- Calculatoarele au implementate această metodă de calcul a expresiilor, expresiile infixate fiind folosite pentru ca oamenii să înţeleagă sensul algebric al acestora.

Algoritmul de evaluare a unei expresii postfixate

Algoritm 21

```
1: while mai există cuvinte în expresie do
2:     citește cuvântul următor
3:     if cuvântul este o valoare then
4:         push (cuvânt)
5:     else cuvântul este un operator
6:         pop() două valori de pe stivă
7:         rezultat = operația aplicată pe cele două valori
8:         push(rezultat)
9:     end if
10: end while
11: rezultat final = pop()
```

Algoritmul 21 nu tratează eventualele erori de *underflow* ce apar de obicei atunci când expresia furnizată nu este validă. Pentru aceasta, va trebui să fie inserate operaţii de verificare dacă stiva este sau nu goală, iar în acest caz să aruncăm o excepţie sau un mesaj.

Exemplu de folosire a algoritmului de evaluare a formei postfixate

Vom evalua expresia infixată 5 + ((1 + 2) * 4) - 3 ce poate fi scrisă sub forma postfixată astfel: 5 1 2 + 4 ∗ + 3 − Mai jos este prezentată secvenţa de paşi prin care este evaluată expresia de mai sus conform algoritmului 21.

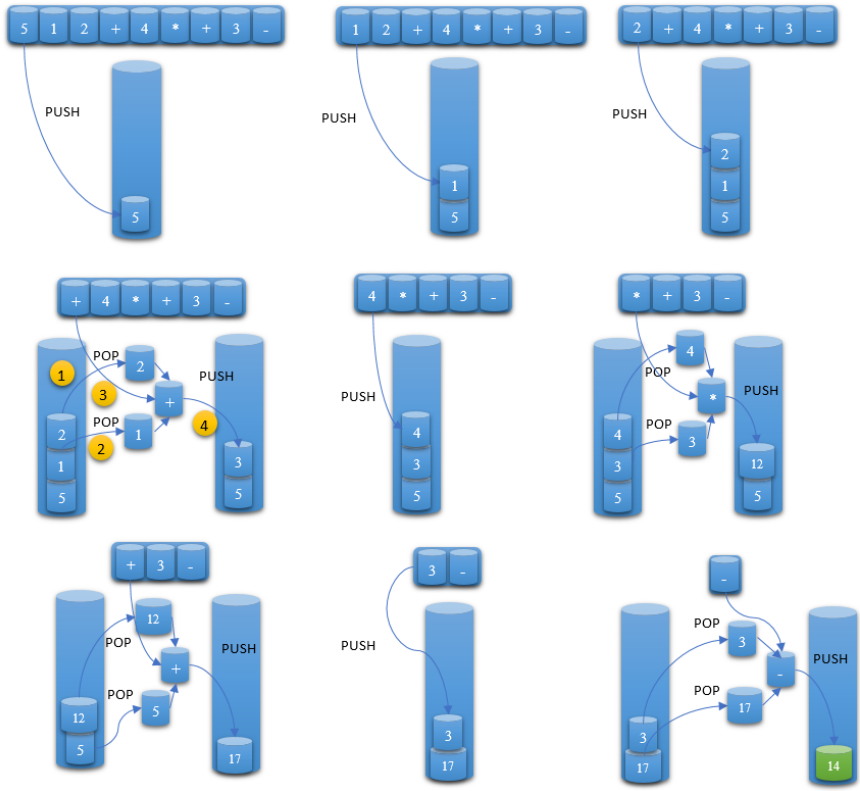


Figura 2.11: Evaluarea expresiei 5 1 2 + 4 ∗ + 3 −.

Cerculeţele de culoare galbenă marchează ordinea în care se execută operaţiile atunci când se întâlneşte un operator în expresie. La final tot ce avem de făcut este un ultim *POP* pentru a obţine rezultatul.

Translatarea unei expresii din forma infixată în forma postfixată

După cum am văzut anterior, forma infixată a fost mai întâi transformată în formă postfixată și după aceea s-a evaluat expresia. Trecerea de la forma infixată la cea infixată are loc conform algoritmului 22.

Algoritm 22

```
1: inițializează stiva
2: while mai există simboluri în expresie do
3:     citește simbolul următor
4:     if simbolul este o ( then
5:         Pune ( în stivă
6:     else
7:         if simbolul este un număr then
8:             Afișează număr
9:         else
10:            if simbol citit este un operator then
11:                if în vârful stivei este un alt operator cu precedența mai mare sau egal cu a acestuia then
12:                    Scoate operator din stivă
13:                    Afișează operator
14:                end if
15:                Pune simbol citit în stivă
16:            else                                ▷ Urmează o paranteză )
17:                Scoate simbolurile din stivă până la întâlnirea unei (
18:                Afișează simbolurile scoase
19:                Scoate ( din stivă
20:            end if
21:        end if
22:    end if
23: end while
24: Scoate și afișează toți operatorii din stivă
```

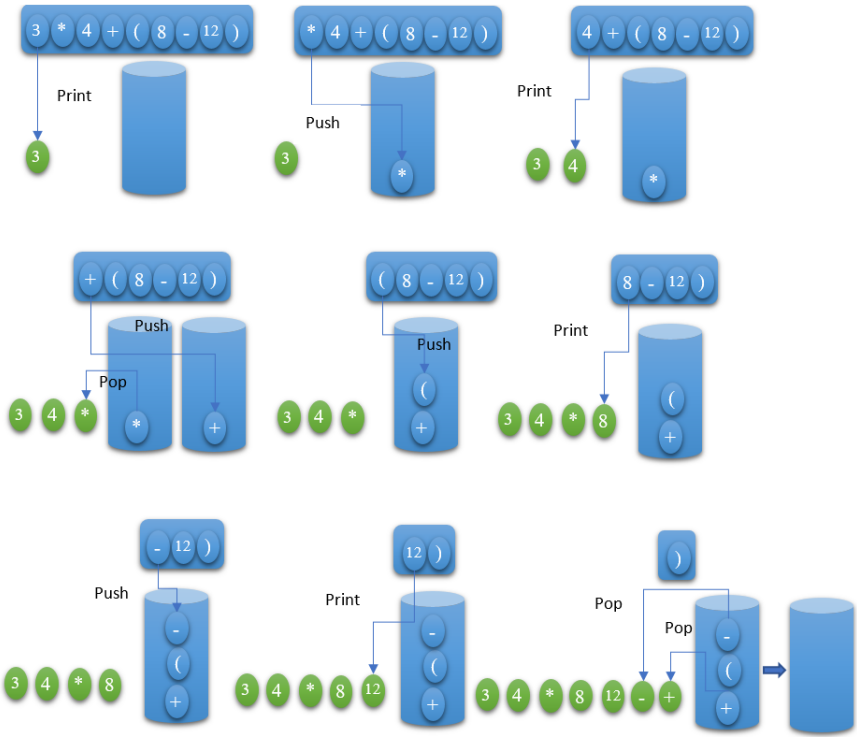


Figura 2.12: Transformarea expresiei 3 * 4 + (8 - 12) în formă postfixată.

În figura 2.12 este reprezentată aplicarea algoritmului 22 pentru expresia 3 * 4 + (8 - 12). Cu verde s-a reprezentat expresia postfixată obținută din cea infixată.

Stiva este folosită pentru a reține temporar operatorii obținuți din expresia infixată.

Expresia obținută (forma postfixată) este 3 4 * 8 12 - +.

2.2.3 Coada

Stiva este o structură de tip FIFO ce poate fi accesată și modificată prin două operațiuni $ENQUEUE$ și $DEQUEUE$. Ca și stiva, această structură poate fi implementată ca un șir sau ca o listă înlănțuită.

Operațiunea de inserare a unui nou elment în coadă se numește $ENQUEUE$ și este defapt operțiunea de inserare a unui element nou la finalul listei, reprezentat de algoritmul 14 și figura 2.6.

Ca și în cazul stivei, operațiunea *DEQUEUE* este defapt operțiunea de ștergere a unui element de la începutul listei, reprezentat de algoritmul 18 și figura 2.8.

Un exemplu clasic de utilizare a cozii este în cadrul problemei *Producător/Consumator*. Presupunem că există elemente sau sarcini ce *produc* o anumită resursă care va trebui *consumată* unul sau mai multe elmente. În cazul în care nu există resursă ce trebuie folosită, consumatorul trebuie să aștepte. Pe de altă parte dacă cel care produce nu are cui să livreze resursa, asemenea va trebui să aștepte.

Pentru a elimina acest timp de așteptare se folosește o coadă așa cum este reprezentată în figura 2.13.

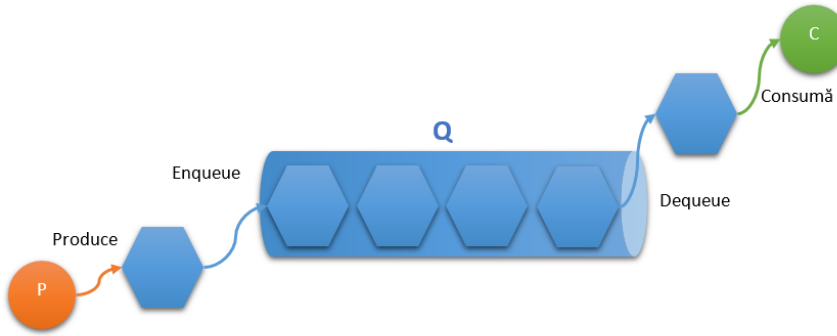


Figura 2.13: Exemplu de utilizare a cozii.

Imediat ce Producătorul creează o nouă resursă o va plasa în coadă și apoi va începe realizarea unei noi resurse. Consumatorul va scoate din coadă elementul de la începutul acesteia și apoi imediat va începe procesul de consumare a resursei. În acest fel coada reține toate resursele în ordinea producerii lor și tot astfel acestea vor fi oferite consumatorului în vederea utilizării acestor resurse. În continuare vom analiza alte structuri de date bazate pe cele studiate până acum și anume șiruri și liste.

2.3 Tabele Hash

Tabelele *Hash* sau *Hash Map* sunt structuri de date ce implementează un dicționar de elemente de tip cheie-valoare. Un element din această structură este format din două părți asociate: cheie și valoare.

Valoarea elementului este stocată în structură la un anumit index, index obținut prin apelul unei funcții de hash având drept cheia ca și parametru. Cheia poate fi de orice tip de dată; de exemplu un șir de caractere sau un întreg, float etc.

De regulă, funcția de hashing este o funcție injectivă ce asignează fiecărei chei un index unic. Există și situații în care se generează pentru chei diferite același index, ceea ce înseamnă o coliziune, situație ce va fi detaliată în continuare.

Pentru a stabili un cadru formal de expunere a problemei vom folosi următoarele notații:

- Un tabel hash de m elemente este $T[0..m-1]$
- k este cheia unui element
- v este valoarea unui element din structură
- U este mulțimea tuturor valorilor pe care le poate lua orice cheie
- h funcția de hashing definită $h : U \to \{0, 1, ..., m-1\}$
- $h(k) = v$ reprezintă valoarea stocată la cheia k

Pentru o mai bună înțelegere a problemei vom alege ca exemplu stocarea datelor dintr-o agendă telefonică. Aici cheile vor fi numele persoanelor iar valorile vor fi numerele de telefon ale acestora.

În figura 2.14 s-a reprezentat această colecție și mai exact interpretarea cheilor de către funcția hash pentru a afla locația la care este stocată valoarea asociată respectivei chei.

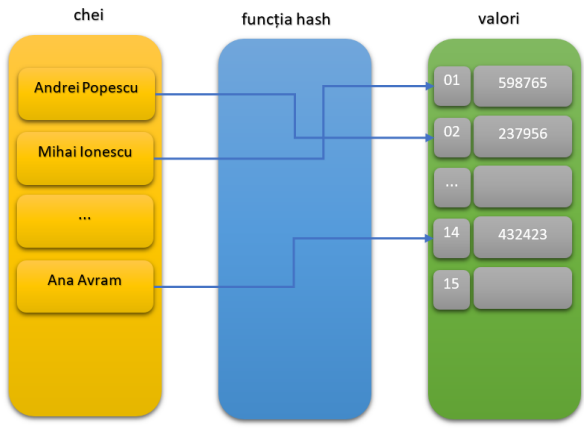


Figura 2.14: Exemplu de utilizare a unei funcții hash.

Există posibilitatea ca pentru două chei diferite funcția h să returneze același index. Altfel spus $h(k_1) = h(k_2)$. Această situație poartă denumirea de *coliziune*. Soluția ideală ar fi să evităm aceste coliziuni. Totuși, când aceste situații apar, pot fi rezolvate prin mai multe metode cum ar fi înlănțuirea sau adresarea deschisă.

2.3.1 Înlănțuirea

În cazul înlănțuirii se pun toate elemente cu aceeași valoare returnată de funcția de hash într-o listă.

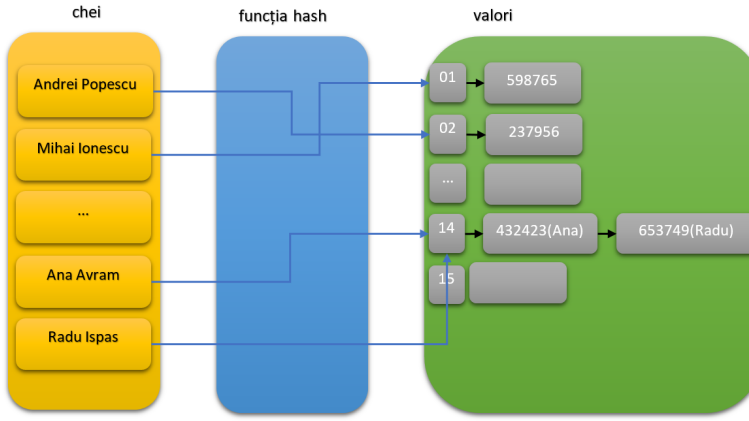


Figura 2.15: Exemplu de coliziune a unei funcții hash.

De exemplu pentru cheile "Ana Avram" și "Radu Ispas" h generează aceeași valoare și anume indexul 14. Aici apare o situație de coliziune. La acest index se va salva o listă cu valorile telefoanelor astfel: pe prima poziție se asociază numărul de telefon pentru "Ana", iar pe a doua poziție cel al lui Radu.

Când se accesează $h(\text{"}AnaAvram\text{"})$ se evaluează h ca fiind 14, se accesează în $O(1)$ lista ce conține cheia căutată. Căutarea se va face prin compararea cheilor inserate în listă cu cheia în cauză.

Analiza înlănțuirii

Cât durează pentru a căuta un element în această structură?

Dat fiind un tabel de hash T cu m poziții și n elemente, factorul de încărcare α este n/m. Dacă fiecare element este stocat doar un index unic, atunci $\alpha = 1$, iar acesta reprezintă cel mai bun scenariu.

În cel mai rău caz cele n sunt sunt legate la același nod și atunci $m = 1$ și evident $\alpha = n$. Timpul de căutare în listă este de $O(n)$ pentru că avem defapt o singură listă înlănțuită.

Performanța depinde de cât de bine reușește funcția de hashing să aloce m chei cât mai diferite.

2.3.2 Funcții de hashing

Ce face ca o funcție de hashing să genereze indecși unici pentru chei unice? Pentru a răspunde la întrebare va trebui euristic să alegem k numere aleatorii distribuite independent și uniform pe intervalul $[0, 1)$ astfel încât funcția $h(k) = \lfloor km \rfloor$ să genereze valori unice.

Majoritatea funcțiilor de hashing pornesc de la presupunerea că valorile cheilor sunt numere naturale $\mathbb{N} = 0, 1, 2, ...$ Adică $h(x) = x$. Dacă o cheie este dată de un caracter atunci se poate alege valoarea ASCII a acelui carater. Dacă o cheie este dată de un șir de caractere atunci indexul se poate afla ca suma valorilor ASCII a caraterelor din șir.

Exemplu șirul "<>" poate fi encodat astfel $60 + 62 = 122$. Problema este că 122 poate fi obținut și din codificarea șirului ";?" adică $59 + 63 = 122$. Vom vedea în cele ce urmează câteva metode de a elimina astfel de coliziuni.

Metoda împărțirii

Metoda divizării folosește restul împărțirii dintre cheie și m - numărul total de chei. Astfel $h(k) = k \bmod m$.

Se recomandă ca mărimea tabelei de hashing, m, să fie un număr impar mai mare decât $n/3$ pentru ca riscul de coliziuni să fie cât mai mic.

Metoda multiplicării

Această medotă conține două etape:

- Se multiplică cheia k cu o constantă $A \in (0, 1)$ și se extrage partea fracțională a lui kA
- Multiplicăm valoarea obținută cu m și reținem partea întreagă a rezultatului

Funcția de hashing $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$.

Un avantaj al acestei metode este că valoarea lui m nu este atât de importantă. m poate fi o putere a lui 2, deoarece operațiunea de multiplicare cu 2 este mai ușor de implementat.

Pentru a proteja cât mai bine de coliziuni, există funcții de hashing care folosesc algoritmi de criptografie pentru a genera valori mari de hashing. Cel mai popular dintre aceștia este SHA-1[5].

2.3.3 Adresarea deschisă

În cadrul adresării deschise toate elementele sunt stocate în cadrul tabelului de hashing. Astfel fiecare element din tabel conține fie un set dinamic fie NULL. Căutarea unui element înseamnă parcurgerea fiecărei adrese și interogarea valorii acelui element.

Nu există liste și niciun element nu este stocat "în afara" tabelului ca în cazul înlănțuirii.

Pentru a efectua operațiunea de inserare, vom examina succesiv locațiile tabelului până când găsim o poziție asignată cu NULL. Acolo vom aloca cheia și asocia valoarea elementului de inserat.

În loc să căutăm în poziții fixe 0,1,..., m-1, căutarea poziției de inserție depinde de cheia ce va fi inserată. Funcția de hashing devine: $h : U \times 0, 1, ..., m - 1 \to 0, 1, ..., m - 1$.

Astfel, pentru orice cheie k secvența $< h(k,0), h(k,1), ... >$ este o permutare a $< 0, 1, ..., m-1 >$ astfel că fiecare poziție a tabelului va fi aleasă până la epuizarea tuturor pozițiilor.

În algoritmul 23 presupunem că toate elementele lui T sunt chei fără valori atașate.

Algoritm 23 Inserarea într-un tabel de hashing

```
1:  procedure HASH-INSERT(T, k)
2:      i ← 0
3:      while i ≠ m do
4:          j ← h(k, i)
5:          if T[j]=NULL then
6:              T[j] ← k
7:                  return j
8:          else
9:              i ← i + 1
10:         end if
11:     end while
12: return "Overflow in hashtable"
13: end procedure
```

Algoritmul "caută" fiecare cheie k și inserează acolo unde poziția încă nu a fost ocupată, adică acolo unde se îndeplinește condiția $T[j] = NULL$.

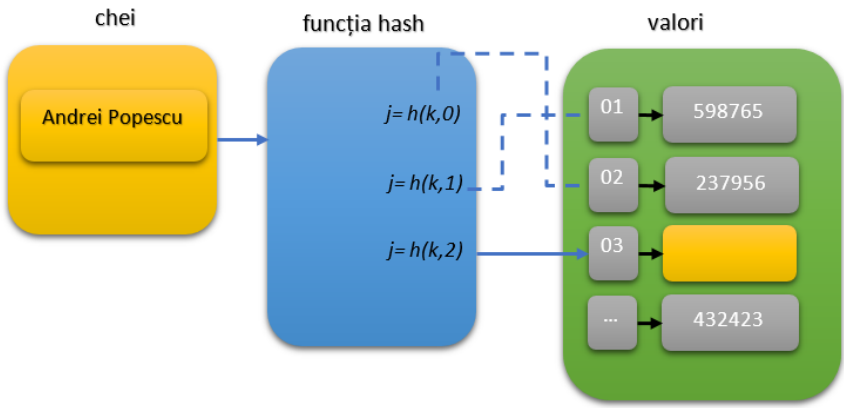


Figura 2.16: Exemplu de inserare a unei chei intr-un tabel de hashing.

Cu linie punctată s-a reprezentat locațiile j unde $T[j] \neq NULL$, unde

nu s-a putut insera valoarea asociată cu cheia k generată de șirul de caractere "Andrei Popescu".

Pentru căutarea unei chei, algoritmul 24 are ca intrare tabela T și cheia k. Aceasta va returna poziția j dacă la această locație a fost găsită cheia k, sau NULL dacă cheia k nu a fost găsită.

Algoritm 24 Căutarea] într-un tabel de hashing

```
1:  procedure HASH-SEARCH(T, k)
2:      i ← 0
3:      j ← h(k, j)
4:      while i ≠ m AND T[j] ≠ NULL do
5:          if T[j]=k then
6:                  return j
7:          end if
8:          i ← i + 1
9:          j ← h(k, i)
10:     end while
11: return NULL
12: end procedure
```

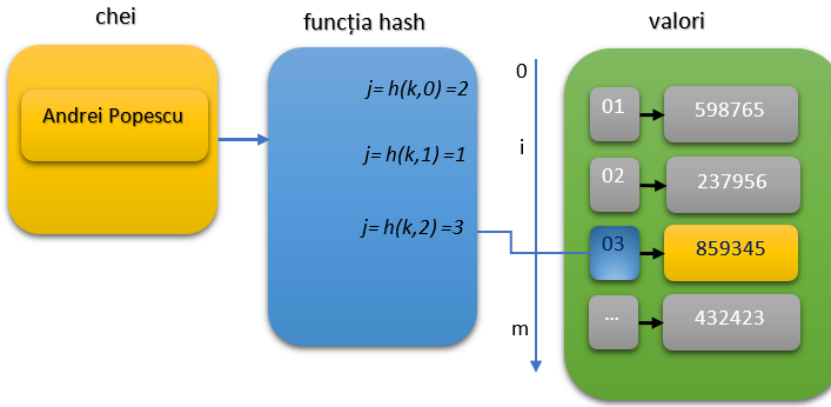


Figura 2.17: Exemplu de căutare a unei poziției corespunzătoare unei chei intr-un tabel de hashing.

Ștergerea unei chei se va face prin marcarea valorii de la locația k a cheii, cu o valoare DELETED în loc de NULL. În felul acesta se face diferența între locații goale (NULL) și locații ce au fost șterse (DELETED).

Analiza adresării deschise

Analiza se bazează ca și în cazul înlănțuirii, pe factorul de încărcare $\alpha = n/m$ cu $n \leq m$ ceea ce înseamnă cu $\alpha \leq 1$. Se presupune că distribuția cheilor este uniformă.

În cazul ideal, secvența $< h(k,0), h(j,1), ..., h(k,m-1) >$ ce este folosită pentru a insera cheia k poate fi orice permutare $< 0, 1, ..., m-1 >$.

Dat fiind factorul de încărcare $\alpha = n/m \leq 1$, numărul de încercări pentru a creea o nouă poziție pentru inserare este de maxim $1/(1-\alpha)$.

Astfel inserarea are loc într-un timp linear. Analog și căutarea va avea loc de asemenea într-un timp linear.

2.3.4 Aplicații ale tabelelor de hashing

Tabelele de hashing sunt des folosite pentru a implementa șiruri asociative în special în limbaje script precum Ruby, Python sau PHP.

De asemenea aceste tabele sunt folosite uneori pentru a stoca indecșii bazelor de date.

Un alt mod de a utiliza tabelele de hashing este în implementarea cache-urilor în diverse aplicații. Astfel se crește timpul de acces la date.

Reprezentarea mulțimilor se realizează foarte ușor cu aceste tabele deoarece cheia este unică, aspect specific mulțimilor.

3 Grafuri

Acest capitol introduce conceptul de graf, care este structura de date fundamentală pentru majoritatea algoritmilor studiați în continuare.

3.1 Noțiuni generale

Un graf este o pereche notată $G =< V, E >$ unde V este mulțimea de vârfuri sau noduri, iar $E \subseteq V \times V$ este mulțimea de muchii.

O muchie de la vârful a la vârful b este notată cu perechea ordonată (a, b). În unele grafuri denumite *grafuri ponderate*, acestei perechi ii se poate asocia o valoare ce poartă denumirea de costul muchiei.

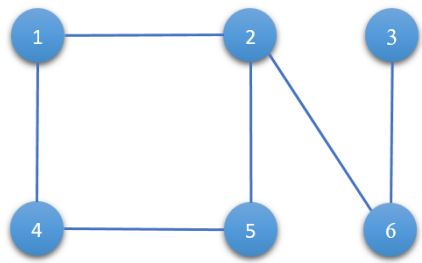
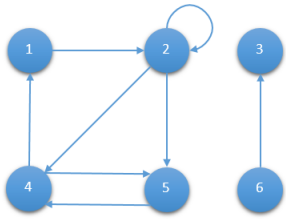


Figura 3.1: Exemplu de graf.

În graful din figura 3.1 $G = (V, E)$ cu $V = 1, 2, 3, 4, 5, 6$ și $E = (1, 2), (2, 1), (1, 4), (4, 1), (2, 5), (5, 2), (4, 5), (5, 4), (2, 6), (6, 2), (3, 6), (6, 3)$.

În cele ce urmează vom defini câteva noțiuni de bază ce vor ajuta în expunerile ulterioare ale algoritmilor ce se bazează pe grafuri.

Un graf orientat sau un digraf, este o pereche notată tot $G =< V, E >$ în care fiecare muchie este o perechea notată u, v unde muchia formată între nodurile u, v pleacă din u şi ajunge în v. În acest tip de graf, este permisă muchia ce are acelaşi nod ca destinaţie şi sursă. În figura 3.2a este prezentat un graf orientat.



(a) Graf orientat

(b) Subgraf obținut din graful orientat

Figura 3.2: Exemplu de graf orientat.

Un *subgraf* este un graf $G_s =< V^{'}, E^{'} >$, unde $V^{'} \subseteq V$ iar $E^{'} \subseteq E$, adică se mențin doar acele muchii ce unesc noduri din $V^{'}$. Un exemplu de subgraf se poate observa în figura 3.2b unde s-au păstrat nodurile 1 și 2 cu muchiile atașate acestor noduri.

Un *graf parțial* este un graf $G_p =< V, E^{''} >$ unde $E^{''} \subseteq E$. Astfel se păstrează numărul de noduri din graful inițial, dar se elimină una sau mai multe muchii. Un exemplu de astfel de graf se găsește în figura 3.3. Acest graf a fost obținut plecând de la graful din figura 3.2a.

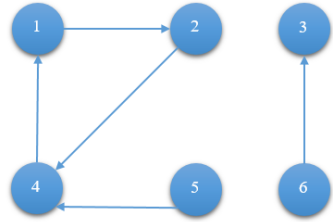


Figura 3.3: Exemplu de graf parțial.

Un graf *neorientat* este un graf $G =< V, E >$ în care ordinea dispunerilor nodurilor în pereche nu contează. O muchie va fi formată din mulțimea u, v unde $u, v \in V$ și $u \neq v$. Prin convenție vom folosin notația (u, v) pentru a delimita o muchie dintr-un graf neorientat. Un astfel de graf este reprezentat în figura 3.1.

Două noduri a și b sunt *adiacente* dacă sunt unite print-o muchie.

Un *drum* este o succesiune de muchii de forma:

$(a_1, a_2), (a_2, a_3), ..., (a_{n-1}, a_n)$

Un exemplu de astfel de drum se găsește în figura 3.4 trasat cu roșu. Drumul poate fi exprimat astfel: (1,2), (2,5), (5,4).

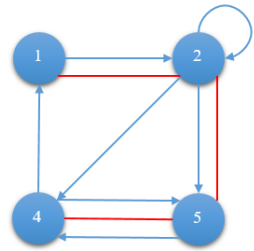


Figura 3.4: Exemplu de graf parțial.

Lungimea drumului este egală cu numărul muchiilor din care este alcătuit.

Un *drum simplu* este acel drum în care niciun vârf nu se repetă.

Un *ciclu* este un drum simplu, cu excepția primului și ultimului vârf care sunt unul și același.

Un *graf aciclic* este un graf ce nu conține niciun ciclu.

Un graf neorientat este *conex* dacă între oricare două vârfuri există un drum. Pentru grafuri orientate, această opțiune este întărită: un graf este *tare* conex dacă între oricare două noduri a și b există cel puțin un drum de la a la b dar și de la b la a.

În figura 3.5 avem un exemplu de graf conex.

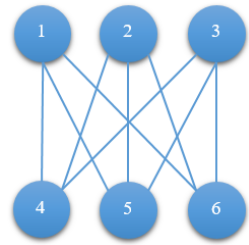


Figura 3.5: Exemplu de graf neorientat conex.

O *componentă conexă* este un subgraf conex minimal, adică un subgraf conex în care niciun vârf din subgraf nu este adiacent cu niciun alt vârf din exteriorul subgrafului. Împărțirea unui graf $G =< V, E >$ în componentele

sașe conexe determină o partiție a lui V și una a lui E. În figura 3.6 avem o reprezentare a unor componente conexe.

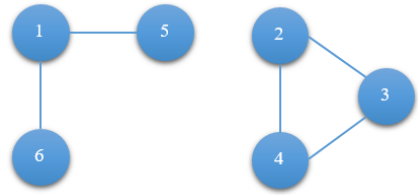


Figura 3.6: Exemplu de graf cu 2 componente conexe: 1,5,6 și 2,3,4.

Vârfurilor unui graf neorientat li se pot atașa informații numite uneori *valori*, iar muchiilor li se pot atașa informații numite *lungimi* sau *costuri*.

3.2 Reprezentarea grafurilor

Există mai multe moduri de a reprezenta un graf, fiecare având avantaje și dezavantaje. Există mai multe criterii conform cărora se alege tipul de reprezentare:

- memoria necesară alocării grafului
- timpul necesar accesării unei muchii din graf
- timpul necesar accesării vecinilor unui nod

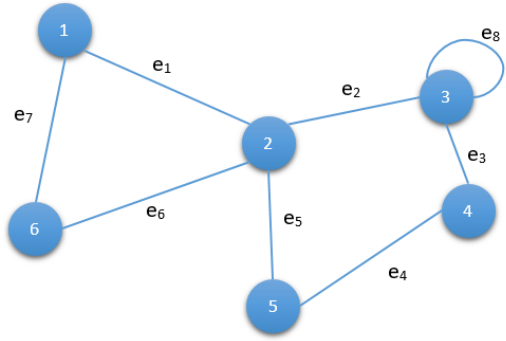


Figura 3.7: Graf neorientat.

Având ca exemplu graful din figura 3.7 vom analiza fiecare tip de reprezentare a grafurilor.

3.2.1 Matricea de adiacență

Se numește matrice de adiacență A, o matrice de $|V| \times |V|$ în care $A[i,j] = true$ dacă nodurile i și j sunt adiacente și $A[i,j] = false$ în caz contrar. O variantă alternativă ar fi să atribuim lui $A[i,j]$ costul muchiei dintre i și j considerând $A[i,j] = +\infty$ atunci când cele două vârfuri nu sunt adiacente. Pentru graful din figura 3.7 matricea de adiacență este cea din tabelul 3.1.

0	1	0	0	0	1
1	0	1	0	1	1
0	1	1	1	0	0
0	0	1	0	1	0
0	1	0	1	0	0
1	1	0	0	0	0

Tabela 3.1: Reprezentarea unei matrice de adiacență

Cu ajutorul matricilor de adiacență putem afla în timp constant dacă o muchie există sau nu în graf. Este suficient să interogăm $A[i,j]$ și vom afla dacă muchia între i și j există.

Există totuși două dezavantaje ale acestei reprezentări. În primul rând spațiul alocat pentru a reprezenta muchiile grafului este $\Theta(V^2)$ chiar dacă graful conține mai puțin de $|V|$ muchii. În al doilea rând, pentru a interoga toți vecinii nodului i va trebui parcursă întreaga linie i, pentru a interoga toate cele $|V|$ noduri, chiar dacă i are un singur vecin.

3.2.2 Liste de muchii

O listă de muchii este un șir de perechi ordonate, fiecare pereche sau tuplu reprezentând o muchie. Dacă unei muchii ii se asociază un cost, atunci perechea devine un triplu format din nodul i, nodul j si costul c.

Pentru graful din figura 3.7, lista de muchii este următoarea:

(1,2,e_1),(2,3,e_2),(3,3,e_8),(3,4,e_3),(4,5,e_4),(5,2,e_5),(6,2,e_6),(1,6,e_7)

unde $e_{1..8}$ reprezintă costul unei muchii adică un număr întreg.

Aceste liste sunt simplu de alcătuit și spațiul necesar alocării lor este liniar: $\Theta(E)$, dar căutarea unei anumite muchii ce leagă nodurile i și j necesită o parcurgere liniară a $|E|$ muchii.

3.2.3 Liste de adiacență

Într-o listă de adiacență fiecare nod i conține un șir de noduri adiacent lui. Astfel avem o listă sau un șir de $|V|$ liste de adiacență precum cea din figura 3.8. Această listă corespunde de asemenea grafului din figura 3.7.

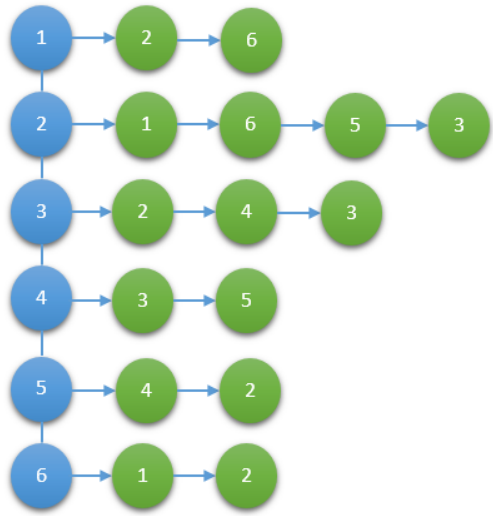


Figura 3.8: Liste de adiacență

Putem accesa lista oricărui nod într-un timp constant. Pentru a afla dacă muchia (i, j) se află sau nu în graf, interogăm lista de adiacență a lui i și căutăm în această listă nodul j. Timpul necesar acestei căutări este $\Theta(d)$ unde d este *gradul* lui i, adică numărul total de vecini al lui i.

Ca și memorie o listă va ocupa cel puțin $|V|$ poziții în cazul în care toate nodurile sunt *izolate* (adică nu au niciun vecin). Dacă fiecare nod are un număr maxim de vecini ($|V| - 1$) atunci memoria alocată va fi în $\Theta(V^2)$.

3.3 Aplicații ale grafurilor

Teoria grafurilor are aplicații în diferite domenii dintre care iată câteva:

- modelarea frecvețelor de tact în circuite digitale
- în construcții pentru estimarea cablajului necesar
- la rutarea semnalelor în rețele predefinite

- la estimarea costurilor drumurilor în hărți
- la evaluarea și predicția evenimentelor economice
- la efectuarea diferitelor simulări de la dinamica consumului până la simularea traficului urban

Acestea sunt doar câteva aplicații concrete ale teoriei grafurilor. Evident lista este mult mai lungă, în continuarea acestui curs urmând să descoperim și alte potențiale utilizări ale algoritmilor ce utilizează grafuri.

4 Arbori

Capitolul prezintă conceptul de arbore și câteva cazuri particulare de arbori.

4.1 Noțiuni generale

Un *arbore* este un graf conex, aciclic. Există mai multe tipuri de arbori pe care le vom detalia în cele ce urmează.

Un arbore *liber* $T(V, E)$ este un graf neorientat conex și fără cicluri.

În figura 4.1 este reprezentat un arbore liber.

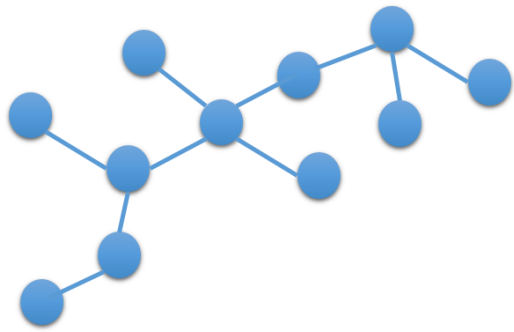


Figura 4.1: Exemplu de arbore liber.

Un arbore are următoarele proprietăți:

1. Conține exact $|V|$ noduri și exact $|V - 1|$ muchii
2. T este aciclic
3. Oricare două noduri sunt conectate printr-un drum unic
4. Ștergerea unei muchii crează doi arbori
5. Adăugarea unei muchii duce la crearea unui ciclu

O *pădure* este un graf format din doi sau mai mulți arbori. În figura 4.2 avem un exemplu de o pădure alcătuită din trei componente conexe, fiecare componentă fiind un arbore.

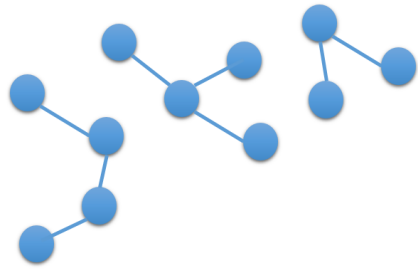


Figura 4.2: Exemplu de pădure.

În figura 4.3 au fost reprezentate două cicluri marcate cu verde. În clipa în care se găsește cel puțin un ciclu, arborele devine un simplu graf conex.

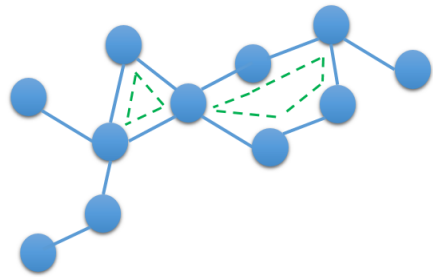


Figura 4.3: Exemplu de graf ce conține ciclu.

Un *arbore cu rădăcină* are următoarele proprietăți:

1. Are un nod aparte numită *rădăcină*
2. Fiecare nod, în afară de rădăcină, are un părinte unic
3. Fiecare nod are 0 sau mai mulți fii.
4. Un nod fără succesori se numește *frunză* sau *terminal*

Un exemplu de arbore cu rădăcină este prezentat în figura 4.4.

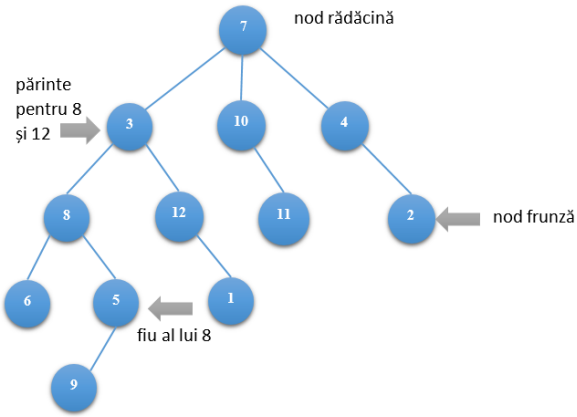


Figura 4.4: Exemplu de arbore cu rădăcină.

Pe lângă proprietățile de mai sus, există o serie de concepte definite pentru un arbore cu rădăcină, concepte care vor fi menționate în algoritmii ce urmează a fi prezentați.

1. Un subarbore al unui arbore este un subgraf conex nevid al acestuia.

2. Un nod v se numește *descendent* al lui u dacă $pred(v) = u$

3. *Înălțimea* unui nod este numărul de muchii de pe cel mai lung drum de la acel nod la un nod frunză. Aceasta se exprimă astfel:

$$h(n) = \begin{cases} 0 & n\ este\ frunză \\ 1 + \max\limits_{k}(h(S_k(n)) & altfel \end{cases} \tag{4.1}$$

   unde $S_k(n)$ este subarborele ce are ca rădăcină fiul k al nodului n

4. *Înălțimea arborelui* este înălțimea rădăcinii

5. *Adâncimea unui nod* este numărul de muchii de la nod la rădăcină. Recursiv, adâncimea se poate defini astfel:

$$A(nod) = A(pred(nod)) + 1,\ A(rădăcină) = 0 \tag{4.2}$$

În figura 4.5 sunt exemplificate câteva din proprietățile ale arborelui cu rădăcină.

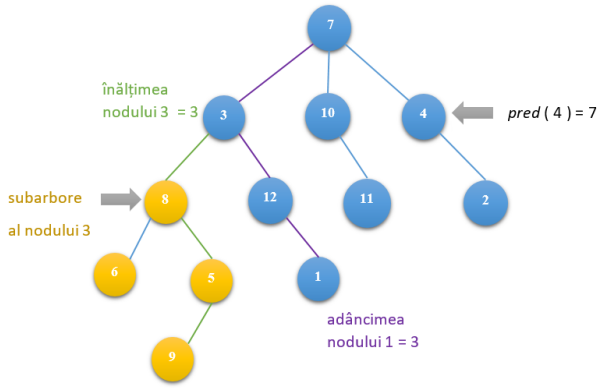


Figura 4.5: Alte proprietăți ale arborelui cu rădăcină.

4.2 Reprezentarea arborilor

În cadrul acestui subcapitol se vor prezenta două dintre implementările posibile arborilor cu rădăcină.

4.2.1 Reprezentarea cu ajutorul tablourilor

Pentru a reprezenta un arbore T cu ajutorul tablourilor va trebui să:

1. Numerotăm nodurile arborelui T de la 1 la n.
2. Asociem nodurilor, elementele șirului astfel: nodului i corespunde locația i din șir.
3. Pe fiecare poziție i se memorează locația părintelui. Astfel $T[i] = j$ unde j este părintele lui i.
4. Dacă $A[i] = 0$ atunci i este rădăcina arborelui.
5. Valorile nodurilor (denumite și chei) se vor salva într-un alt șir V pe pozițiile i corespunzătoare nodurilor din arborele T. Astfel în $C[i]$ se reține valoarea din nodul i.

Se poate evident optimiza structura de mai sus, menținând un singur șir T. Pe fiecare poziție din tablou păstrăm un obiect format din două informații: locația părintelui și cheia asociată nodului curent.

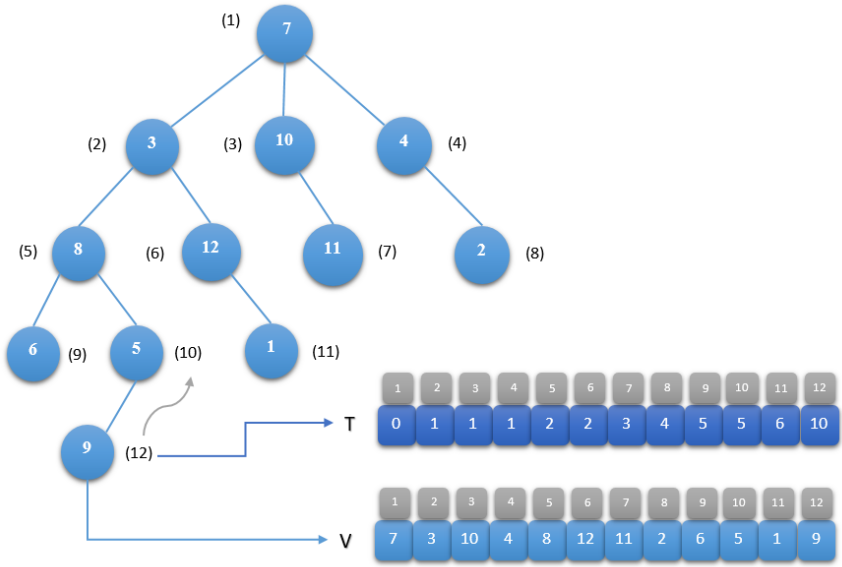


Figura 4.6: Alte proprietăți ale arborelui cu rădăcină.

În figura 4.6 se găsește reprezentarea arborilor folosind șirul T pentru păstrarea pozițiilor părinților nodurilor și V pentru păstrarea cheilor sau valorilor din interiorul nodurilor.

Acest tip de reprezentare "fiu către părinte" are avantajul că accesul la părintele unui nod se face în $O(1)$. Totodată parcurgerea arborelui în direcția părintelui pornind de la nodul curent, se realizează într-un timp $O(a)$, unde a este adâncimea nodului.

Un dezavantaj al acestui tip de structură este că nu permite adăugarea sau ștergerea unui nod în/din arbore, tocmai pentru că un șir este imuabil.

Dificultatea utilizării acestei implementări devine clară atunci când, dat fiind un nod n, se dorește aflarea fiilor acestuia sau a înălțimii sale. Totodată ordinea în care fii unui nod sunt dispuși în arbore nu este bine definită.

Se poate impune o regulă ce să rezolve problema ordonării fiilor unui nod: "Pozițiile asociate fiilor unui nod, cresc de la stânga spre dreapta. Astfel nu este imperativ ca fii să fie plasați pe poziții consecutive în șir".

4.2.2 Reprezentarea cu ajutorul listelor simplu înlănțuite

Pornind de la reprezentarea grafurilor cu ajutorul listelor vom genera pentru fiecare nod o listă a fiilor săi.

În lista principală vom păstra nodurile din arbore în ordinea următoare: de la rădăcină către fii și de la stânga la dreapta.

Vârfurile ce au cel puțin un fiu vor reține o listă cu fii lor. Nodurile frunză vor indica un element NULL. Totodată în fiecare nod din listă va conține și informația despre cheia acelui nod.

În figura 4.7 este reprezentat sub forma unor liste de vecini, arborele din figura 4.4.

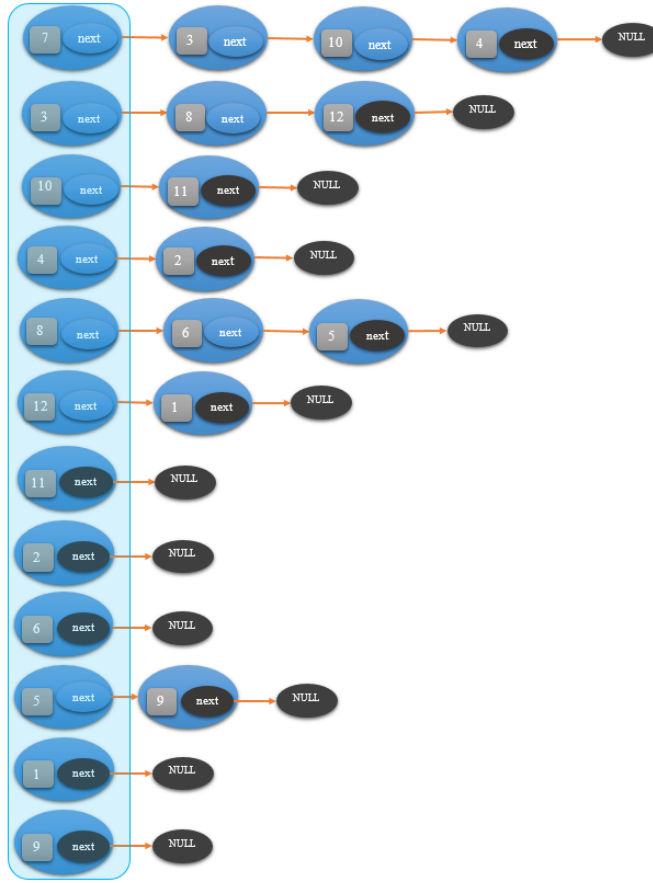


Figura 4.7: Reprezentarea ca lista de vecini a arborelui.

Avantajele acestei organizări a arborilor sunt evident de parcurgerile dinspre părinte către fii, de la rădăcină spre frunze. Pentru aceasta este nevoie de un timp liniar $O(d)$ unde d sunt vecinii unui nod n. Dezavantajul constă în aflarea drumului în sens invers, adică de la fiu către părinte. Pentru aceasta, trebuie interogată întreaga listă de noduri, fiecare nod interogând vecinii săi. Timpul necesar acestei operații este unul pătratic și anume $O(dn)$. Acest neajuns poate fi ușor reparat dacă lucrăm cu liste dublu sau multiplu înlănțuite (după tipul de arbore cu rădăcină).

Acest tip de reprezentare este foarte potrivit pentru arbori orientați și anume acei arbori în care muchiile au un sens, întotdeauna de la rădăcină către frunze.

4.2.3 Reprezentarea cu ajutorul listelor multiplu înlănțuite

În cazul în care se dorește parcurgerea arborilor cât mai eficient în orice sens, și de la rădăcină spre frunze și invers, putem folosi o structură asemănătoare cu listele dublu înlănțuite care conține pe lângă valoarea (cheia) nodului două referințe:

1. Referința către părinte. În cazul în care nodul este chiar rădăcina, referința va fi NULL

2. Listă de referințe către copii. Dacă nodul este frunză atunci lista este formată dintr-un obiect NULL

O reprezentare în pseudocod a unui nod al arborelui poate fi cea de mai jos:

```
struct {
Data v;
Node parinte;
Lista<Node> fii;
} Node;
```

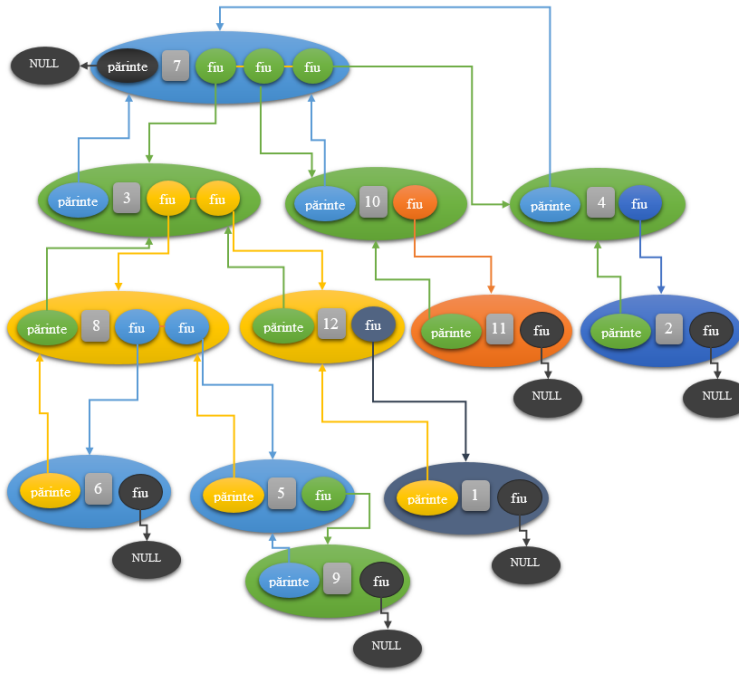


Figura 4.8: Reprezentarea ca listă multiplu înlănțuită, a arborelui.

Fiind reprezentat printr-o o listă, arborele trebuie să răspundă la următoarele operațiuni:

- Inserare
- Modificare
- Căutare/Parcurgere
- Ștergere

Astfel inserarea unui nod înseamnă refacerea legăturilor noului nod cu cele ale nodului în care va fi inserat.

Spre exemplu dacă vrem să adăugăm un nou nod cu valoarea 17 ca fiu al nodului 4 atunci există 3 legături ce trebuie creeate (marcate cu roșu în figura 4.9).

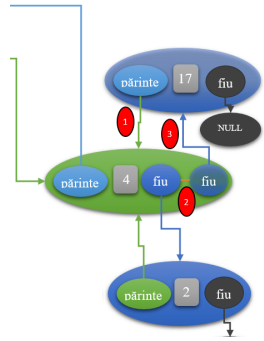


Figura 4.9: Inserarea unui nou nod într-un arbore cu rădăcină.

După această operație nodul 4 va avea doi fii, în ordinea lor de la stânga la dreapta 2 și 17.

Algoritmul pentru inserare a unui nou nod este 25.

Algoritm 25 Inserarea unui nou nod în arbore

```
1:  procedure INSERT_NEW(T, cheie, nouaval)
2:      ▷ Alocăm memorie pentru noul nod
3:      newnode ← new Node
4:          ▷ Atribuim valoarea specifică acestuia
5:      newnode.v ← nouaval
6:          ▷ Nodul nou nu are fii
7:      newnode.fii ← NULL
8:          ▷ Parcurgem arborele până la nodul părinte al nodului de inserat
9:      node ← SEARCHTREE_BYKEY(T.RĂDĂCINĂ, CHEIE)()
10:         ▷ Refacem cele două legături conectând astfel nodul nou la arbore
11:             ▷ De la noul nod la părinte
12:     newnode.parinte = node
13:         ▷ Și de la părinte la nou nod
14:     list ← node.fii
15:     while list.next ≠ NULL do
16:         list ← list.next
17:     end while
18:     lis.next ← newnode
19: end procedure
```

Algoritm 26 Căutarea unui nod în arbore după cheie

```
1: procedure SearchTree_ByKey(node, cheie)
2:     ▷ Se preia lista de copii ai nodului curent
3:     list ← node.fii
4:     while list.next ≠ NULL do
5:   ▷ Dacă se găsește cheia în lista de fii, se returnează acel fiu
6:         if list.v = cheie then return list
7:         end if
8:       ▷ Dacă nu, se parcurge în adâncime arborele
9:         newnode ← SearchTree_ByKey(list, cheie)()
10:        if newnode.v = cheie then return newnode
11:        end if
12:      ▷ Se trece la următorul fiu din listă
13:        list ← list.next
14:    end while
15: end procedure
```

În continuare vom insista (cu o excepție în cazul *heap-urilor*) pe acest tip de reprezentare deoarece traversarea arborelui, în ambele sensuri, se realizează în mod optim.

5 Greedy

5.1 Exemplu introductiv

Începem studiul algoritmilor cu următoarea problemă de sortare

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

6 Divide et impera

6.1 The First Section

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

7 Programare dinamica

7.1 The First Section

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

8 String Matching

8.1 The First Section

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

9 Back Tracking

9.1 The First Section

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

10 Metode iterative

10.1 The First Section

Here is some copy for your book[1].

Notice how this figure is in black and white when the book.tex is formatted, but color in the eBook and PDFs.

Here is a simple indexed term. And here I use the permuted index operator Indexed Thingy, although you can also have a different index name if you need it: Indexed Thingy2.

An Unnumbered Section

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Another Unnumbered Section

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum

dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Here we'll illustrate some more indexing commands. **A Company Name** will be added to the Index (see zzInit.tex) as both the name alone, and under the Companies index section. You can also do *My Movie Name* as a file, and *The Cool Project* project names. Heres a final end note[2], and one more citation [2].

Notes

Capitolul 1 — Introducere

Capitolul 2 — Structuri de date de bază

Capitolul 3 — Grafuri

Capitolul 4 — Arbori

Capitolul 5 — Greedy

[1]Here is a footnote about this copy

[2]This is the last one

Capitolul 6 — Divide et impera

[1]Here is a footnote about this copy

[2]This is the last one

Capitolul 7 — Programare dinamica

[1]Here is a footnote about this copy

[2]This is the last one

Capitolul 8 — String Matching

[1]Here is a footnote about this copy

[2]This is the last one

Capitolul 9 — Back Tracking

[1]Here is a footnote about this copy

[2]This is the last one

Capitolul 10 — Metode iterative

[1]Here is a footnote about this copy

[2]This is the last one

Bibliografie

[1] William B. Gragg and Martin H. Gutknecht. *Stable Look-Ahead Versions of the Euclidean and Chebyshev Algorithms*, pages 231–260. Birkhäuser Boston, Boston, MA, 1994.

[2] Tomas Pfister, J Charles, and A Zisserman. Large-scale Learning of Sign Language by Watching TV (Using Co-occurrences). *Proceedings of the British Machine Vision Conference*, pages 1–11, 2013.

[3] Chris R. Reid, David J. T. Sumpter, and Madeleine Beekman. Optimisation in a natural system: Argentine ants solve the towers of hanoi. *Journal of Experimental Biology*, 214(1):50–58, 2010.

[4] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[5] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. *Finding Collisions in the Full SHA-1*, pages 17–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

Lista cu autorii citați

Lista cu sursele imaginilor

Note: Additional information on sources and attributions for illustrations and figures can be found online at http://YourBooksURL.com/

All Abbreviations and Terms

Nomenclature

Abbreviations (Computation)

Abbreviations (Neuroscience)

.