Week 5: String Algorithms, Approximation, Randomisation, Ethics

Strings

Strings

A *string* is a sequence of characters.

An *alphabet* Σ is the set of possible characters in strings.

Examples of strings:

- C program
- HTML document
- DNA sequence
- Digitised image

Examples of alphabets:

- ASCII
- Unicode
- {0,1}
- {A,C,G,T}

... Strings

Notation:

- *length(P)* … #characters in *P*
- λ … *empty* string (*length*(λ) = 0)
- Σ^m … set of all strings of length *m* over alphabet Σ
- Σ^* … set of all strings over alphabet Σ

νω denotes the *concatenation* of strings ν and ω

Note: length(νω) = length(ν)+length(ω) λω = ω = ωλ

... Strings

Notation:

- *substring* of *P* … any string *Q* such that $P = \nu Q \omega$, for some $\nu,\omega \in \Sigma^*$
- *prefix* of *P* … any string *Q* such that $P = Q\omega$, for some $\omega \in \Sigma^*$
- *suffix* of *P* … any string *Q* such that $P = \omega Q$, for some $\omega \in \Sigma^*$

Exercise #1: Strings

The string `a/a` of length 3 over the ASCII alphabet has

- how many prefixes?
- how many suffixes?
- how many substrings?

- 4 prefixes: `""` `"a"` `"a/"` `"a/a"`
- 4 suffixes: `"a/a"` `"/a"` `"a"` `""`
- 6 substrings: `""` `"a"` `"/"` `"a/"` `"/a"` `"a/a"`

Note:
`""` means the same as λ (= empty string)

... Strings

ASCII (American Standard Code for Information Interchange)

- Specifies mapping of 128 characters to integers 0..127
- The characters encoded include:
  - upper and lower case English letters: A-Z and a-z
  - digits: 0-9
  - common punctuation symbols
  - special non-printing characters: e.g. *newline* and *space*

Ascii	Char	Ascii	Char	Ascii	Char	Ascii	Char
0	Null	32	Space	64	@	96	`
1	Start of heading	33	!	65	A	97	a
2	Start of text	34	"	66	B	98	b
3	End of text	35	#	67	C	99	c
4	End of transmit	36	$	68	D	100	d
5	Enquiry	37	%	69	E	101	e
6	Acknowledge	38	&	70	F	102	f
7	Audible bell	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Horizontal tab	41	)	73	I	105	i
10	Line feed	42	*	74	J	106	j
11	Vertical tab	43	+	75	K	107	k
12	Form feed	44	,	76	L	108	l
13	Carriage return	45	-	77	M	109	m
14	Shift in	46	.	78	N	110	n
15	Shift out	47	/	79	O	111	o
16	Data link escape	48	0	80	P	112	p
17	Device control 1	49	1	81	Q	113	q
18	Device control 2	50	2	82	R	114	r
19	Device control 3	51	3	83	S	115	s
20	Device control 4	52	4	84	T	116	t
21	Neg. acknowledge	53	5	85	U	117	u
22	Synchronous idle	54	6	86	V	118	v
23	End trans. block	55	7	87	W	119	w
24	Cancel	56	8	88	X	120	x
25	End of medium	57	9	89	Y	121	y
26	Substitution	58	:	90	Z	122	z
27	Escape	59	;	91	[	123	{
28	File separator	60	<	92	\	124	|
29	Group separator	61	=	93	]	125	}
30	Record separator	62	>	94	^	126	~
31	Unit separator	63	?	95	_	127	Forward del.

... Strings

UTF-8

- Most common Unicode standard
- Specifies mapping of around 150,000 characters
- ASCII-compatible
  - `0b0xxxxxxx` ASCII letters 0-127
- Two, three and four byte long characters
  - `0b110xxxxxxxxxxx` most Latin-script alphabets
  - `0xExxxxx` Chinese, Japanese, Korean characters
  - `0xFxxxxxxx` mathematical symbols, emojis

unicode.org/emoji/charts/full-emoji-list.html

Pattern Matching

Example (pattern checked *backwards*):

- *Text* … `abacaab`
- *Pattern* … `abacab`

... Pattern Matching

Given two strings T (text) and P (pattern),
the *pattern matching problem* consists of finding a substring of T equal to P

Applications:

- Text editors
- Search engines
- Biological research

... Pattern Matching

Naive pattern matching algorithm

- checks for each possible shift of P relative to T
  - until a match is found, or
  - all placements of the pattern have been tried

```
NaiveMatching(T,P):
|  Input  text T of length n, pattern P of length m
|  Output starting index of a substring of T equal to P
|         -1 if no such substring exists
|
|  for all i=0..n-m do
|  |  j=0                          // check from left to right
|  |  while j<m and T[i+j]=P[j] do  // test i^th shift of pattern
|  |  |  j=j+1
|  |  |  if j=m then
|  |  |     return i               // entire pattern checked
|  |  |  end if
|  |  end while
|  end for
|  return -1                       // no match found
```

Analysis of Naive Pattern Matching

Naive pattern matching runs in O(n·m)

Examples of worst case (forward checking):

- T = `aaa…ah`
- P = `aaah`
- may occur in DNA sequences
- unlikely in English text

Exercise #2: Naive Matching

Suppose all characters in P are different.

Can you accelerate `NaiveMatching` to run in $O(n)$ on an n-character text T?

When a mismatch occurs between $P[j]$ and $T[i+j]$, shift the pattern all the way to align $P[0]$ with $T[i+j]$

⇒ each character in T checked at most twice

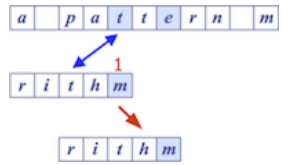
Example:

```
abcdabcdeabcc   abcdabcdeabcc
abcde                abcde
```

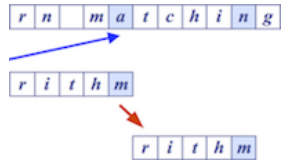
Boyer-Moore Algorithm

The *Boyer-Moore* pattern matching algorithm is based on two heuristics:

- *Looking-glass heuristic*: Compare P with subsequence of T moving *backwards*
- *Character-jump heuristic*: When a mismatch occurs at $T[i]$=`c`
  - if P contains `c` ⇒ shift P so as to align the last occurrence of `c` in P with $T[i]$

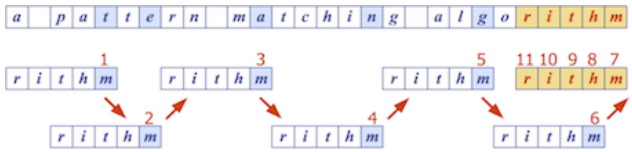


  - otherwise ⇒ shift P so as to align $P[0]$ with $T[i+1]$ (a.k.a. "big jump")



... Boyer-Moore Algorithm

Example:



... Boyer-Moore Algorithm

Boyer-Moore algorithm preprocesses pattern P and alphabet Σ to build

- *last-occurrence function* L
  - L maps Σ to integers such that $L(c)$ is defined as
    - the largest index i such that $P[i]=c$, or
    - -1 if no such index exists

Example: Σ = {a,b,c,d}, P = acab

c	a	b	c	d
$L(c)$	2	3	1	-1

- L can be represented by an array indexed by the numeric codes of the characters
- L can be computed in $O(m+s)$ time (m … length of pattern, s … size of Σ)

... Boyer-Moore Algorithm


```
BoyerMooreMatch(T,P,Σ):
|  Input  text T of length n, pattern P of length m, alphabet Σ
|  Output starting index of a substring of T equal to P
|         -1 if no such substring exists
|
|  L=lastOccurenceFunction(P,Σ)
|  i=m-1, j=m-1                  // start at end of pattern
|  repeat
|  |  if T[i]=P[j] then
|  |  |  if j=0 then
|  |  |     return i             // match found at i
|  |  |  else
|  |  |     i=i-1, j=j-1         // keep comparing
|  |  |  end if
|  |  else                       // character-jump
|  |  |  i=i+m-min(j,1+L[T[i]])
|  |  |  j=m-1
|  |  end if
|  until i≥n
|  return -1                     // no match
```

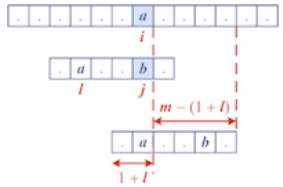
- Biggest jump (m characters ahead) occurs when $L[T[i]] = -1$

... Boyer-Moore Algorithm

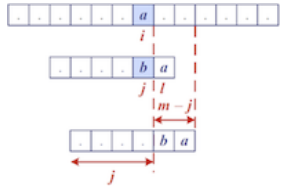
l … last occurrence L[T[i]] of character T[i]

Why $i = i + m - min(j,1+l)$?

- Case 1: $1+l \le j \quad \Rightarrow i = i+m-(1+l)$



- Case 2: $j \le l \quad \Rightarrow i = i+m-j$

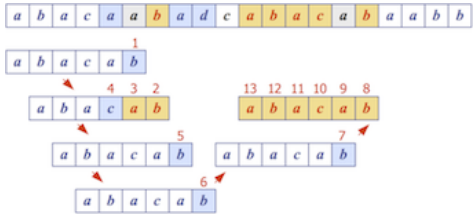


Exercise #3: Boyer-Moore algorithm

For the alphabet Σ = {a,b,c,d}

1. compute last-occurrence function L for pattern P = **abacab**
2. trace Boyer-More on P and text T = abacaabadcabacabaabb
   - how many comparisons are needed?

c	a	b	c	d
$L(c)$	4	5	3	-1



13 comparisons in total

... Boyer-Moore Algorithm

Analysis of Boyer-Moore algorithm:

- Runs in $O(nm+s)$ time
  - m … length of pattern n … length of text s … size of alphabet
- Example of worst case:
  - T = aaa … a
  - P = baaa
- Worst case may occur in images and DNA sequences but unlikely in English texts

⇒ Boyer-Moore significantly faster than naive matching on English text

---

Knuth-Morris-Pratt Algorithm

The *Knuth-Morris-Pratt* algorithm …

- compares the pattern to the text *left-to-right*
- but shifts the pattern more intelligently than the naive algorithm

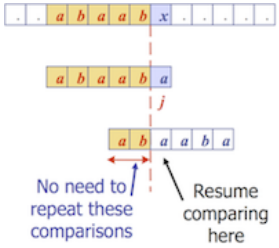
Reminder:

- Q is a *prefix* of P … $P = Q\omega$, for some $\omega \in \Sigma^*$
- Q is a *suffix* of P … $P = \omega Q$, for some $\omega \in \Sigma^*$

... Knuth-Morris-Pratt Algorithm

When a mismatch occurs …

- what is the most we can shift the pattern to avoid redundant comparisons?
- Answer: the largest *prefix* of *P[0..j-1]* that is a *suffix* of *P[1..j-1]*



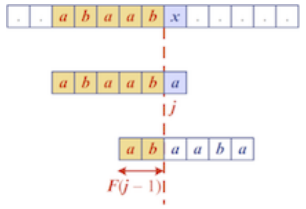
... Knuth-Morris-Pratt Algorithm

KMP preprocesses the pattern *P[0..m-1]* to find matches of its prefixes with itself

- *Failure function F(j)* defined as
  - the size of the *largest prefix* of *P[0..j]* that is also a *suffix* of *P[1..j]*
    - for each position *j*=0..*m-1*
- if mismatch occurs at P_j ⇒ advance *j* to *F(j-1)*

Example: *P* = `abaaba`

j	0	1	2	3	4	5
P_j	a	b	a	a	b	a
F(j)	0	0	1	1	2	3



... Knuth-Morris-Pratt Algorithm

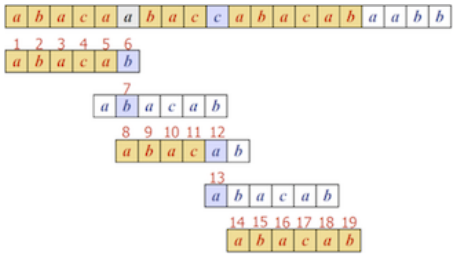

```
KMPMatch(T,P):
|  Input  text T of length n, pattern P of length m
|  Output starting index of a substring of T equal to P
|         -1 if no such substring exists
|
|  F=failureFunction(P)
|  i=0, j=0                    // start from left
|  while i<n do
|  |  if T[i]=P[j] then
|  |  |  if j=m-1 then
|  |  |     return i-j         // match found at i-j
|  |  |  else
|  |  |     i=i+1, j=j+1       // keep comparing
|  |  |  end if
|  |  else if j>0 then          // mismatch and j>0?
|  |  |  j=F[j-1]               //  → shift pattern to i-F[j-1]
|  |  else                     // mismtach and j still 0?
|  |  |  i=i+1                 //  → begin at next text character
|  |  end if
|  end while
|  return -1                   // no match
```

Exercise #4: KMP-Algorithm

1. compute failure function *F* for pattern *P* = `abacab`
2. trace Knuth-Morris-Pratt on *P* and text *T* = `abacaabaccabacabaabb`
   - how many comparisons are needed?

---

j	0	1	2	3	4	5
P_j	a	b	a	c	a	b
F(j)	0	0	1	0	1	2



19 comparisons in total

... Knuth-Morris-Pratt Algorithm

Analysis of Knuth-Morris-Pratt algorithm:

- Failure function can be computed in *O(m)* time (→ next slide)
- At each iteration of the while-loop, either
  - *i* increases by one, or
  - the pattern is shifted ≥1 to the right ("shift amount" *i-j* increases since always F(j-1)<j)

  ⇒ *i* can be incremented at most *n* times, pattern can be shifted at most *n* times

  ⇒ there are no more than 2·*n* iterations of the while-loop

⇒ KMP's algorithm runs in *optimal time O(m+n)*

... Knuth-Morris-Pratt Algorithm

Construction of the failure function matches pattern against *itself*:

```
failureFunction(P):
|  Input  pattern P of length m
|  Output failure function for P
|
|  F[0]=0                  // F[0] is always 0
|  j=1, len=0
|  while j<m do
|  |  if P[j]=P[len] then
|  |     len=len+1         // we have matched len+1 characters
|  |     F[j]=len          // P[0..len-1] = P[len-1..j]
|  |     j=j+1
|  |  else if len>0 then   // mismatch and len>0?
|  |     len=F[len-1]      //  → use already computed F[len] for new len
|  |  else                 // mismatch and len still 0?
|  |     F[j]=0            //  → no prefix of P[0..j] is also suffix of P[1..j]
|  |     j=j+1             //  → continue with next pattern character
|  |  end if
|  end while
|  return F
```

Exercise #5:

Trace the `failureFunction` algorithm for pattern *P* = **abaaba**

```
                          ⇒         F[0]=0
j=1, len=0, P[1]≠P[0] ⇒         F[1]=0
j=2, len=0, P[2]=P[0] ⇒ len=1, F[2]=1
j=3, len=1, P[3]≠P[1] ⇒ len=F[0]=0
j=3, len=0, P[3]=P[0] ⇒ len=1, F[3]=1
j=4, len=1, P[4]=P[1] ⇒ len=2, F[4]=2
j=5, len=2, P[5]=P[2] ⇒ len=3, F[5]=3
```

... Knuth-Morris-Pratt Algorithm

Analysis of failure function computation:

- At each iteration of the while-loop, either
  - *i* increases by one, or
  - the "shift amount" *i-j* increases by at least one (remember that always F(j-1)<j)
- Hence, there are no more than 2·*m* iterations of the while-loop

⇒ failure function can be computed in *O(m)* time

Boyer-Moore vs KMP

Boyer-Moore algorithm

- decides how far to jump ahead based on the mismatched character in the text
- works best on large alphabets and natural language texts (e.g. English)

Knuth-Morris-Pratt algorithm

- uses information embodied in the pattern to determine where the next match could begin
- works best on small alphabets (e.g. `A,C,G,T`)

For the keen: The article "Average running time of the Boyer-Moore-Horspool algorithm" shows that the time is inversely proportional to size of alphabet

---

Word Matching With Tries

Preprocessing Strings

Preprocessing the *pattern* speeds up pattern matching queries

- After preprocessing *P*, KMP algorithm performs pattern matching in time proportional to the text length

If the text is large, immutable and searched for often (e.g., works by Shakespeare)

- we can preprocess the *text* instead of the pattern

Tries

... Tries

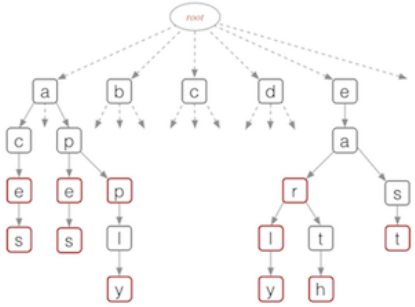
A *trie* …

- is a tree-like data structure
- for compact representation of a set of strings
  - e.g. all the words in a text, a dictionary etc.

Note: Trie comes from *retrieval*, but is pronounced like "try" to distinguish it from "tree"

Tries

Tries are trees organised using parts of keys (rather than whole keys)



Exercise #6:

How many words are encoded in the trie on the previous slide?

11

... Tries

Each node in a trie …

- contains one part of a key (typically one character)
- may have up to 26 children
- may be tagged as a "finishing" node
- but even "finishing" nodes may have children

Depth d of trie = length of longest key value

Cost of searching *O(d)* (independent of *n*)

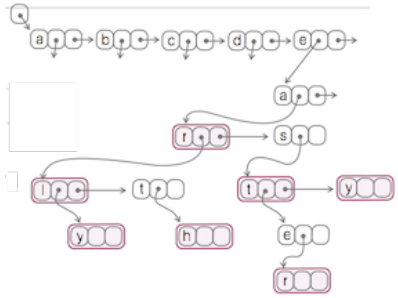
... Tries

Possible trie representation:

```
#define ALPHABET_SIZE 26

typedef struct Node *Trie;

typedef struct Node {
   bool finish;      // last char in key?
   Item data;        // no Item if !finish
   Trie child[ALPHABET_SIZE];
} Node;

typedef char *Key;
```

... Tries

Note: Can also use BST-like nodes for more space-efficient implementation of tries

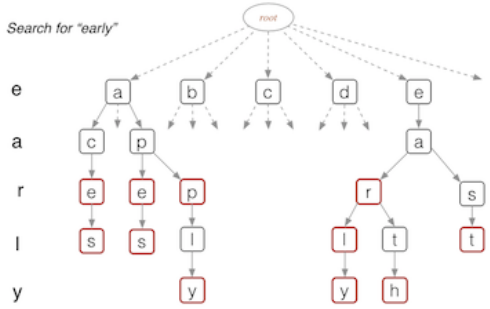


Trie Operations

Basic operations on tries:

1. search for a key
2. insert a key

... Trie Operations



... Trie Operations

Traversing a path, using char-by-char from Key:

```
find(trie,key):
|  Input  trie, key
|  Output pointer to element in trie if key found
|         NULL otherwise
```

```
|
|  node=trie
|  for each char in key do
|  |  if node.child[char] exists then
|  |     node=node.child[char]  // move down one level
|  |  else
|  |     return NULL
|  |  end if
|  end for
|  if node.finish then          // "finishing" node reached?
|     return node
|  else
|     return NULL
|  end if
```

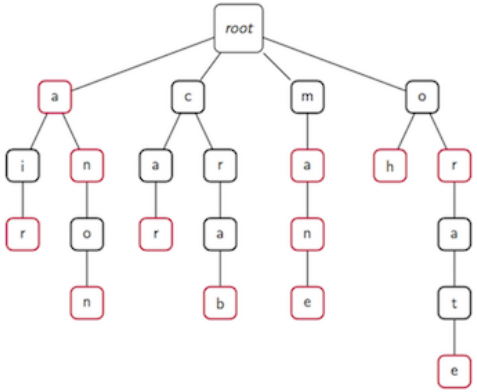
... Trie Operations

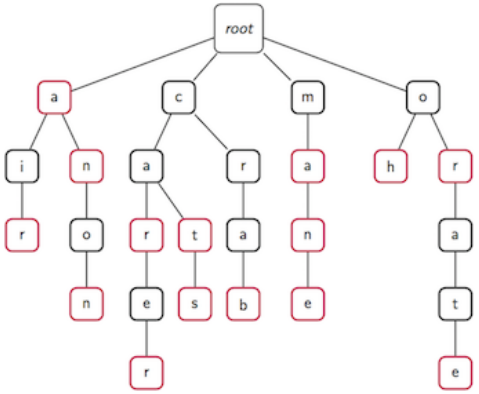
Insertion into Trie:

```
insert(trie,item,key):
|  Input  trie, item with key of length m
|  Output trie with item inserted
|
|  if trie is empty then
|     t=new trie node
|  end if
|  if m=0 then
|     t.finish=true, t.data=item
|  else
|     t.child[key[0]]=insert(t.child[key[0]],item,key[1..m-1])
|  end if
|  return t
```

Exercise #7: Trie Insertion

Insert cat, cats and carer into this trie:





... Trie Operations

Analysis of standard tries:

- *O(n)* space
- insertion and search in time *O(m)*
  - *n* … total size of text (e.g. sum of lengths of all strings in a given dictionary)
  - *m* … size of the string parameter of the operation (the "key")

Word Matching With Tries

Word Matching with Tries

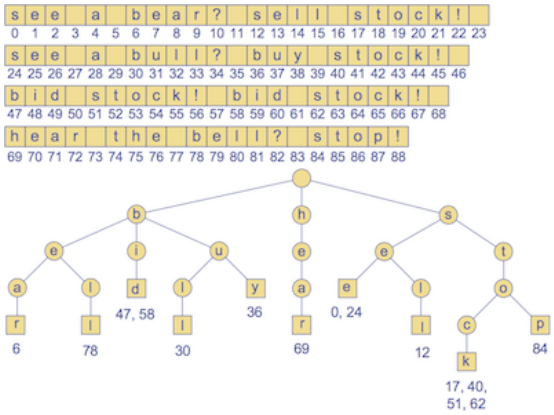
Preprocessing the text:

1. Insert all searchable words of a text into a trie
2. Each finishing node stores the occurrence(s) of the associated word in the text

---

... Word Matching with Tries

Example text and corresponding trie of searchable words:

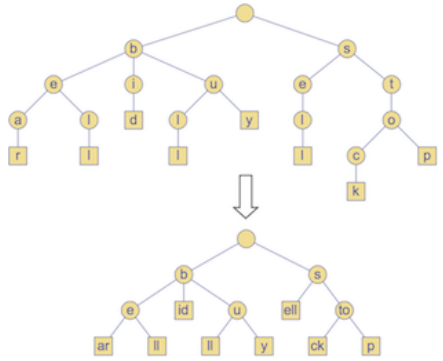


Compressed Tries

Compressed tries …

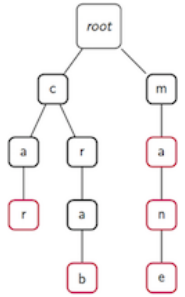
- have internal nodes of degree at least 2 (i.e. non-finishing nodes must have ≥ 2 children)
- are obtained from standard tries by compressing "redundant" chains of nodes

Example:



Exercise #8: Compressed Tries

Consider this uncompressed trie:



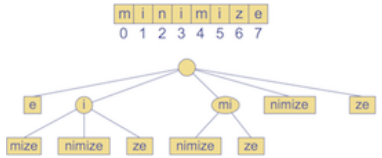
How many nodes (including the root) are needed for the compressed trie?

7

Pattern Matching With Suffix Tries

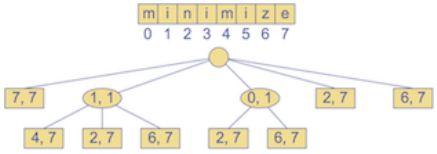
The *suffix trie* of a text T is the compressed trie of all the suffixes of T

Example:



... Pattern Matching With Suffix Tries

Compact representation:



... Pattern Matching With Suffix Tries

Input:

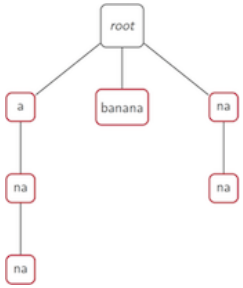
- compact suffix trie for text T
- pattern P

Goal:

- find starting index of a substring of T equal to P

Exercise #9: Suffix Tries

Construct the compressed suffix trie for T = **banana**



... Pattern Matching With Suffix Tries


```
suffixTrieMatch(trie,P):
|  Input  compact suffix trie for text T, pattern P of length m
|  Output starting index of a substring of T equal to P
|         -1 if no such substring exists
|
|  j=0, v=root of trie
|  repeat
|  |  // we have matched j characters
|  |  if ∃w∈children(v) such that P[j]=T[start(w)] then
|  |  |  i=start(w)            // start(w) is the start index of w
|  |  |  x=end(w)-i+1          // end(w) is the end index of w
|  |  |  if m≤x then   // length of suffix ≤ length of the node label?
|  |  |     if P[j..j+m-1]=T[i..i+m-1] then
|  |  |        return i-j      // match at i-j
|  |  |     else
|  |  |        return -1       // no match
|  |  |  else if P[j..j+x-1]=T[i..i+x-1] then
|  |  |     j=j+x, m=m-x       // update suffix start index and length
|  |  |     v=w                // move down one level
|  |  |  else return -1        // no match
|  |  |  end if
|  |  else
|  |     return -1
|  |  end if
|  until v is leaf node
|  return -1                   // no match
```

... Pattern Matching With Suffix Tries

Analysis of pattern matching using suffix tries:

Suffix trie for a text of size n …

- can be constructed in $O(n)$ time (most efficient implementation)
- uses $O(n)$ space
- supports pattern matching queries in $O(m)$ time
  - m … length of the pattern

Text Compression

Text Compression

Problem: Efficiently encode a given string X by a smaller string Y

Applications:

- Save memory and/or bandwidth

Huffman's algorithm

- computes frequency $f(c)$ for each character c
- encodes high-frequency characters with short code
- no code word is a prefix of another code word
- uses optimal *encoding tree* to determine the code words

... Text Compression

Code … mapping of each character to a binary code word

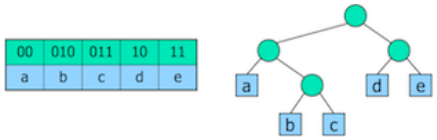
Prefix code … binary code such that no code word is prefix of another code word

Encoding tree …

- represents a prefix code
- each leaf stores a character
- code word given by the path from the root to the leaf (0 for left child, 1 for right child)

... Text Compression

Example:



... Text Compression

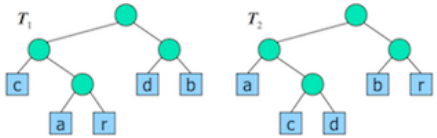
Text compression problem

Given a text T, find a prefix code that yields the shortest encoding of T

- short codewords for frequent characters
- long code words for rare characters

Exercise #10:

Two different prefix codes:



Which code is more efficient for T = `abracadabra`?

T_1 requires 29 bits to encode text T,
T_2 requires 24 bits.

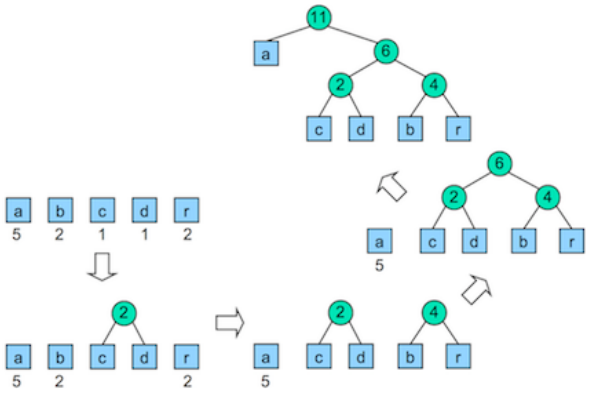
`01011011010000101001011011010` vs `001011000100001100101100`

Huffman Code

Huffman's algorithm

- computes frequency *f(c)* for each character
- successively combines pairs of lowest-frequency characters to build encoding tree "bottom-up"

Example: `abracadabra`



... Huffman Code

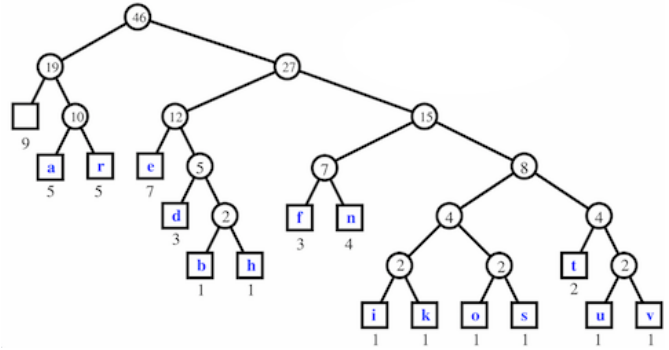
Huffman's algorithm using priority queue:

```
HuffmanCode(T):
|  Input  string T of size n
|  Output optimal encoding tree for T
|
|  compute frequency array
|  Q=new priority queue
|  for all characters c do
|  |  T=new single-node tree storing c
|  |  join(Q,T) with frequency(c) as key
|  end for
|  while |Q|≥2 do
|  |  f1=Q.minKey(), T1=leave(Q)
|  |  f2=Q.minKey(), T2=leave(Q)
|  |  T=new tree node with subtrees T1 and T2
|  |  join(Q,T) with f1+f2 as key
|  end while
|  return leave(Q)
```

Exercise #11: Huffman Code

Construct a Huffman tree for: a fast runner need never be afraid of the dark

Character		a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
Frequency	9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1



... Huffman Code

Analysis of Huffman's algorithm:

- $O(n+d \cdot \log d)$ time
  - n … length of the input text T
  - d … number of distinct characters in T

Approximation

Approximation for Numerical Problems

Approximation is often used to solve numerical problems by

- solving a simpler, but much more easily solved, problem
- where this new problem gives an approximate solution
- and refine the method until it is "accurate enough"

Examples:

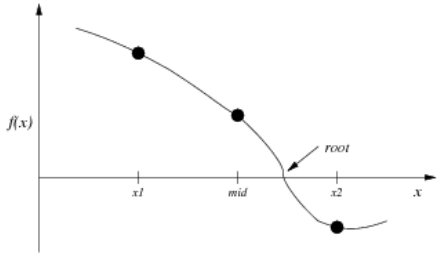
- roots of a function f
- length of a curve determined by a function f

- … and many more

... Approximation for Numerical Problems

Example: Finding Roots

Find where a function crosses the x-axis:



Generate and test: move x_1 and x_2 together until "close enough"

... Approximation for Numerical Problems

A simple approximation algorithm for finding a root in a given interval:

```
bisection(f,x1,x2):
|  Input  function f, interval [x1,x2]
|  Output x∈[x1,x2] with f(x)≅0
|
|  repeat
|  |  mid=(x1+x2)/2
|  |  if f(x1)*f(mid)<0 then
|  |     x2=mid      // root to the left of mid
|  |  else
|  |     x1=mid      // root to the right of mid
|  |  end if
|  until f(mid)=0 or x2-x1<ε   // ε: accuracy
|  return mid
```

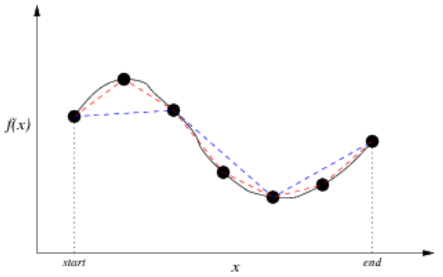
`bisection` guaranteed to converge to a root if f continuous on $[x_1,x_2]$ and $f(x_1)$ and $f(x_2)$ have opposite signs

---

... Approximation for Numerical Problems

Example: Length of a Curve

Estimate length: approximate curve as sequence of straight lines.




```
length=0, δ=(end-start)/StepSize
for each x∈[start+δ,start+2δ,..,end] do
   length = length + sqrt(δ² + (f(x)-f(x-δ))²)
end for
```

Approximation for Problems in NP

Approximation is often used for problems in NP…

- computing a near-optimal solution
- in polynomial time

Examples:

- vertex cover of a graph
- subset-sum problem

Vertex Cover

Reminder: Graph $G = (V,E)$

- set of vertices V
- set of edges E

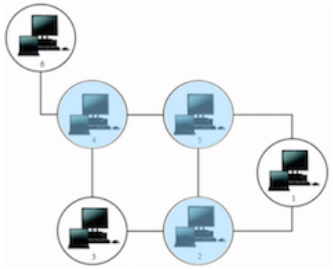
Vertex cover C of G …

- $C \subseteq V$
- for all edges $(u,v) \in E$ either $v \in C$ or $u \in C$ (or both)

⇒ All edges of the graph are "covered" by vertices in C

... Vertex Cover

Example (6 nodes, 7 edges, 3-vertex cover):



Applications:

- Computer Network Security
  - compute minimal set of routers to cover all connections
- Biochemistry

... Vertex Cover

size of vertex cover C … $|C|$ (number of elements in C)

optimal vertex cover … a vertex cover of minimum size

Theorem.
Determining whether a graph has a vertex cover of a given size k is an NP-complete problem.

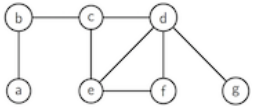
... Vertex Cover

An approximation algorithm for vertex cover:

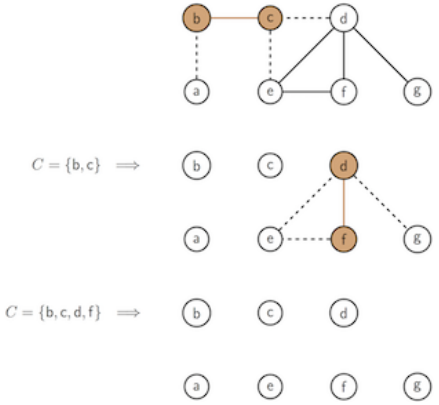
```
approxVertexCover(G):
|  Input  undirected graph G=(V,E)
|  Output vertex cover of G
|
|  C=∅
|  E'=copy of E
|  while E'≠∅ do
|  |  choose any (v,w) ∈ E'
|  |  C = C∪{v,w}
|  |  E' = E' \ {all edges incident on v or w}
|  end while
|  return C
```

Exercise #12: Vertex Cover

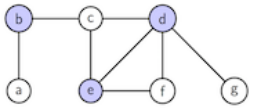
Show how the approximation algorithm produces a vertex cover on:



Possible result:



What would be an optimal vertex cover?



... Vertex Cover

Theorem.
The approximation algorithm returns a vertex cover *at most twice the size* of an optimal cover.

Proof. Any (optimal) cover must include at least one endpoint of each chosen edge.

Cost analysis …

- repeatedly select an edge from E
  - add endpoints to C
  - delete all edges in E covered by endpoints

Time complexity: O(V+E) (adjacency list representation)

Randomised Algorithms

Randomised Algorithms

Algorithms employ randomness to

- improve worst-case runtime
- compute correct solutions to hard problems more efficiently but with low probability of failure
- compute approximate solutions to hard problems

Randomness

Randomness is also useful

- in computer games:
  - may want aliens to move in a random pattern
  - the layout of a dungeon may be randomly generated
  - may want to introduce unpredictability
- in physics/applied maths:
  - carry out simulations to determine behaviour
    - e.g. models of molecules are often assume to move randomly
- in testing:
  - *stress test* components by bombarding them with random data
  - random data is often seen as *unbiased data*
    - gives average performance (e.g. in sorting algorithms)
- in cryptography

Sidetrack: Random Numbers

Reminder: A computer cannot pick a number at random.

Software can only produce *pseudo random numbers*.

- a pseudo random number is one that is predictable
  - (although it may appear unpredictable)

⇒ Implementation may deviate from expected theoretical behaviour

---

... Sidetrack: Random Numbers

The most widely-used technique is called the *Linear Congruential Generator (LCG)*

- it uses a recurrence relation:
  - $X_{n+1} = (a \cdot X_n + c) \bmod m$, where:
    - m is the "modulus"
    - a, $0 < a < m$ is the "multiplier"
    - c, $0 \le c \le m$ is the "increment"
    - X_0 is the "*seed*"
  - if c=0 it is called a *multiplicative congruential generator*

LCG is not good for applications that need extremely high-quality random numbers

- the period length is too short
- *period length* … length of sequence at which point it repeats itself
- a short period means the numbers are correlated

... Sidetrack: Random Numbers

Trivial example:

- for simplicity assume c=0
- so the formula is $X_{n+1} = a \cdot X_n \bmod m$
- try a=11=X_0, m=31, which generates the sequence:

```
11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25,
27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5,
24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1,
11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27,
18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16,
21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10, 17, 1, 11, 28,
29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14, 30, 20, 3, 2, 22, 25, 27, 18,
12, 8, 26, 7, 15, 10, 17, 1, ...
```

- all the integers from 1 to 30 are here
- period length = 30

... Sidetrack: Random Numbers

Another trivial example:

- again let c=0
- try a=12=X_0 and m=30
  - that is, $X_{n+1} = 12 \cdot X_n \bmod 30$
  - which generates the sequence:

```
12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6,
12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6, ...
```

- notice the period length (= 4) … clearly a terrible sequence

---

... Sidetrack: Random Numbers

It is a complex task to pick good numbers. A bit of history:

Lewis, Goodman and Miller (1969) suggested

- $X_{n+1} = 7^5 \cdot X_n \bmod (2^{31}-1)$
- note:
  - 7^5 is 16807
  - 2^{31}-1 is 2147483647
  - $X_0 = 0$ is not a good seed value

Most compilers use LCG-based algorithms that are slightly more involved; see www.mscs.dal.ca/~selinger/random/ for details (including a short C program that produces the exact same pseudo-random numbers as `gcc` for any given seed value)

... Sidetrack: Random Numbers

- Two functions are required:

```
srand(unsigned int seed) // sets its argument as the seed

rand() // uses a LCG technique to generate random
       // numbers in the range 0 .. RAND_MAX
```

  where the constant `RAND_MAX` is defined in `stdlib.h`
  (depends on the computer: on the CSE network, RAND_MAX = 2147483647)

- The period length of this random number generator is very large
  approximately $16 \cdot ((2^{31}) - 1)$

... Sidetrack: Random Numbers

To convert the return value of `rand()` to a number between 0 .. RANGE

- compute the remainder after division by RANGE+1

Using the remainder to compute a random number is not the best way:

- can generate a 'better' random number by using a more complex division
- but good enough for most purposes

Some applications require more sophisticated, *cryptographically secure* pseudo random numbers

---

Exercise #13: Random Numbers

Write a program to simulate 10,000 rounds of Two-up.

- Assume a $10 bet at each round
- Compute the overall outcome and average per round

```
#include <stdlib.h>
#include <stdio.h>

#define RUNS 10000
#define BET  10

int main(void) {
   srand(1234567);      // choose arbitrary seed
   int coin1, coin2, n, sum = 0;
   for (n = 0; n < RUNS; n++) {
      do {
         coin1 = rand() % 2;
         coin2 = rand() % 2;
      } while (coin1 != coin2);
      if (coin1==1 && coin2==1)
         sum += BET;
      else
         sum -= BET;
   }
   printf("Final result: %d\n", sum);
   printf("Average outcome: %f\n", (float) sum / RUNS);
   return 0;
}
```

... Sidetrack: Random Numbers

Seeding

There is one significant problem:

- every time you run a program with the same seed, you get exactly the same sequence of 'random' numbers
  (why?)

To vary the output, can give the random seeder a starting point that varies with time

- an example of such a starting point is the current time, ***time(NULL)***
  (NB: this is different from the UNIX command `time`, used to measure program running time)

```
#include <time.h>
time(NULL) // returns the time as the number of seconds
           // since the Epoch, 1970-01-01 00:00:00 +0000
```

```
// time(NULL) on July 31st, 2020, 12:59pm was 1596164340
// time(NULL) about a minute later was 1596164401
```

Randomised Algorithms

Analysis of Randomised Algorithms

Math needed for the analysis of randomised algorithms:

Sample space… $\Omega = \{\omega_1,\ldots,\omega_n\}$
Probability… $0 \le P(\omega_i) \le 1$
Event… $E \subseteq \Omega$

- Basic probability theory
  - $P(E) = \Sigma_{\omega\in E}\, P(\omega)$
  - $P(\Omega) = 1$
  - $P(\textit{not } E) = P(\Omega\backslash E) = 1 - P(E)$
  - $P(E_1 \textit{ and } E_2) = P(E_1\cap E_2) = P(E_1)\cdot P(E_2)$ if E_1, E_2 *independent*
- Expectation
  - event E has probability $p \Rightarrow$ average number of trials needed to see E is $1/p$
- Combinatorics
  - number of ways to choose k objects from n objects…

$$\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{1\cdot 2\cdot\ldots\cdot k}$$

Exercise #14: Basic Probability

Consider $\Omega = \{$`HH`, `HT`, `TH`, `TT`$\}$ (each outcome with probability ¼)

1. E_1 = first coin lands on heads. What is $P(E_1)$?
2. E_2 = second coin lands on tails. What is $P(E_2)$?
3. Are E_1, E_2 independent?
4. Probability of not (E_1 and E_2)?
5. On average, how often do you have to toss the pair of coins to obtain `HH` or `TT`?

---

1. ½
2. ½
3. Yes
4. 1 – ¼·¼ = ¾
5. 2 times

   Note that $2 = \frac{1}{\frac{1}{2}}$ is the infinite sum $\frac{1}{2}\cdot 1 + \left(1-\frac{1}{2}\right)\cdot\frac{1}{2}\cdot 2 + \left(1-\frac{1}{2}\right)^2\cdot\frac{1}{2}\cdot 3 + \left(1-\frac{1}{2}\right)^3\cdot\frac{1}{2}\cdot 4 + \ldots$

... Analysis of Randomised Algorithms

Randomised algorithm to find *some* element with key k in an unordered array:

```
findKey(L,k):
|  Input  array L, key k
|  Output some element in L with key k
```

```
|
|  repeat
|     randomly select e∈L
|  until key(e)=k
|  return e
```

... Analysis of Randomised Algorithms

Analysis:

- p … ratio of elements in L with key k (e.g. $p = \frac{1}{3}$)
- *Probability of success*: 1 (if $p > 0$)
- *Expected runtime*: $\frac{1}{p}$
  - Example: a third of the elements have key $k \Rightarrow$ expected number of iterations = 3

... Analysis of Randomised Algorithms

If we cannot guarantee that the array contains any elements with key k …

```
findKey(L,k,d):
|  Input  array L, key k, maximum #attempts d
|  Output some element in L with key k
|
|  repeat
|  |  if d=0 then
|  |     return failure
|  |  end if
|  |  randomly select e∈L
|  |  d=d-1
|  until key(e)=k
|  return e
```

... Analysis of Randomised Algorithms

Analysis:

- p … ratio of elements in L with key k
- d … maximum number of attempts
- *Probability of success*: 1 - $(1\text{-}p)^d$
- *Expected runtime*: $\left(\sum_{i=1..d-1} i \cdot (1-p)^{i-1} \cdot p \right) + d \cdot (1-p)^{d-1}$
  - *O(1)* if d is a constant

Randomised Quicksort

Quicksort applies divide and conquer to sorting:

- Divide
  - pick a *pivot* element
  - move all elements smaller than the *pivot* to its left
  - move all elements greater than the *pivot* to its right
- Conquer
  - sort the elements on the left
  - sort the elements on the right

Non-randomised Quicksort

Divide ...

```
partition(array,low,high):
|  Input  array, index range low..high
|  Output selects array[low] as pivot element
|         moves all smaller elements between low+1..high to its left
|         moves all larger elements between low+1..high to its right
|         returns new position of pivot element
|
|  pivot_item=array[low], left=low+1, right=high
|  repeat
|  |  right = find index of rightmost element <= pivot_item
|  |  left  = find index of leftmost element > pivot_item    // left=right if none
|  |  if left<right then
|  |     swap array[left] with array[right]
|  |  end if
|  until left≥right
|  if low<right then
|     swap array[low] with array[right]       // right is final position for pivot
|  end if
|  return right
```

... Non-randomised Quicksort

... and Conquer!

```
Quicksort(array,low,high):
|  Input  array, index range low..high
|  Output array[low..high] sorted
|
|  if high > low then    // termination condition low >= high
|  |   pivot = partition(array,low,high)
|  |   Quicksort(array,low,pivot-1)
|  |   Quicksort(array,pivot+1,high)
|  end if
```

... Non-randomised Quicksort


```
3  6  5  2  4  1    // swap a[left=1] and a[right=5]

3  1  5  2  4  6    // swap a[left=2] and a[right=3]

3  1  2  5  4  6    // swap pivot and a[right=2]

2  1   | 3 |   5  4  6

1  2   | 3 |   5  4  6
```

```
1  2   | 3 |   4   | 5 |   6
```

Worst-case Running Time

Worst case for Quicksort occurs when the pivot is the unique minimum or maximum element:

- One of the intervals `low..pivot-1` and `pivot+1..high` is of size *n-1* and the other is of size 0
  ⇒ running time is proportional to $n + n\text{-}1 + \ldots + 2 + 1$
- Hence the worst case for non-randomised Quicksort is $O(n^2)$

```
1  2  3  4  5  6

1  |  2  3  4  5  6

1  |  2  |  3  4  5  6

            ...

1  |  2  |  3  |  4  |  5  |  6
```

Randomised Quicksort


```
partition(array,low,high):
|  Input  array, index range low..high
|  Output randomly select a pivot element from array[low..high]
|         moves all smaller elements between low..high to its left
|         moves all larger elements between low..high to its right
|         returns new position of pivot element
|
|  randomly select pivot_index∈[low..high]
|  pivot_item=array[pivot_index], swap array[low] with array[pivot_index]
|  left=low+1, right=high
|  repeat
|  |  right = find index of rightmost element <= pivot_item
|  |  left  = find index of leftmost element > pivot_item    // left=right if none
|  |  if left<right then
|  |     swap array[left] with array[right]
|  |  end if
|  until left≥right
|  if low<right then
|     swap array[low] with array[right]       // right is final position for pivot
|  end if
|  return right
```

... Randomised Quicksort

Analysis:

- Consider a recursive call to `partition()` on an index range of size *s*
  - *Good call*:
    both `low..pivot-1` and `pivot+1..high` shorter than ¾·*s*
  - *Bad call*:
    one of `low..pivot-1` or `pivot+1..high` greater than ¾·*s*
- Probability that a call is good: 0.5
  (because half the possible pivot elements cause a good call)

Example of a bad call:

```
6  3  7  5  8  2  4  1

4  3  6  5  1  2   | 7 |   8
```

Example of a good call:

```
4  3  6  5  1  2   | 7 |   8

1  2   | 3 |   5  6  4   | 7 |   8
```

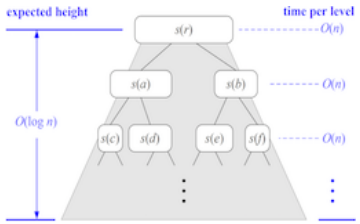
... Randomised Quicksort

n … size of array

From probability theory we know that the expected number of coin tosses required in order to get k heads is 2·k

- For a recursive call at depth *d* we expect
  - *d*/2 ancestors are good calls
    ⇒ size of input sequence for current call is $\leq (¾)^{d/2} \cdot n$
- Therefore,
  - the input of a recursive call at depth $2 \cdot \log_{4/3} n$ has expected size 1
    ⇒ the expected recursion depth thus is *O(log n)*
- The total amount of work done at all the nodes of the same depth is *O(n)*

Hence the expected runtime is *O(n·log n)*



Minimum Cut Problem

Given:

- undirected graph *G=(V,E)*

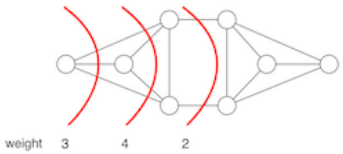
Cut of a graph …

- a partition of *V* into $S \cup T$
  - *S*,*T* disjoint and both non-empty
- its *weight* is the number of edges between *S* and *T*:
  $$\omega(S,T) = |\{\{s,t\} \in E : s \in S, t \in T\}|$$

Minimum cut problem … find a cut of *G* with minimal weight

... Minimum Cut Problem

Example:

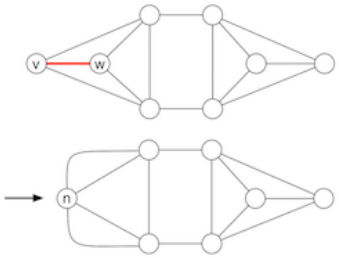


Contraction

Contracting edge $e = \{v,w\}$ …

- remove edge e
- replace vertices v and w by new node n
- replace all edges $\{x,v\}, \{x,w\}$ by $\{x,n\}$

… results in a *multigraph* (multiple edges between vertices allowed) Example:



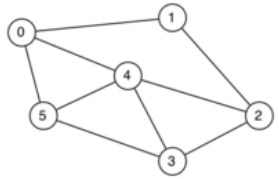
... Contraction

Randomised algorithm for *graph contraction* = repeated edge contraction until 2 vertices remain

```
contract(G):
|   Input  graph G = (V,E) with |V|≥2 vertices
|   Output cut of G
|
|   while |V|>2 do
|      randomly select e∈E
|      contract edge e in G
|   end while
|   return the only cut in G
```

Exercise #15: Graph Contraction

Apply the contraction algorithm twice to the following graph, with different random choices:



... Contraction

Analysis:

V … number of vertices

- *Fact*. Probability of `contract` to result in a minimum cut:
$$\geq 1 / \binom{V}{2}$$
- This is much higher than the probability of picking a minimum cut at random, which is
$$\leq \binom{V}{2} / \left(2^{V-1} - 1\right)$$
because every graph has 2^{V-1}-1 cuts and
  - *Fact*. At most $\binom{V}{2}$ cuts can have minimum weight
- Single edge contraction can be implemented in $O(V)$ time on an adjacency-list representation ⇒ total running time: $O(V^2)$

  (Best known implementation of graph contraction uses $O(E)$ time)

Karger's Algorithm

Idea: Repeat random graph contraction several times and take the best cut found

```
MinCut(G):
|   Input  graph G with V≥2 vertices
|   Output smallest cut found
|
|   min_weight=∞, d=0
|   repeat
|   |   cut=contract(G)
|   |   if weight(cut)<min_weight then
|   |      min_cut=cut, min_weight=weight(cut)
|   |   end if
|   |   d=d+1
|   until d > binomial(V,2)·ln V
|   return min_cut
```

... Karger's Algorithm

Analysis:

V … number of vertices
E … number of edges

- *Probability of success*: $\geq 1 - \frac{1}{V}$

- probability of not finding a minimum cut when the contraction algorithm is repeated $d = \binom{V}{2} \cdot \ln V$ times:

$$\leq \left[1 - 1/\binom{V}{2}\right]^d \leq \frac{1}{e^{\ln V}} = \frac{1}{V}$$

- Total running time: $O(E \cdot d) = O(E \cdot V^2 \cdot log\ V)$
  - assuming graph contraction implemented in $O(E)$

Sidetrack: Maxflow and Mincut

Given: flow network $G=(V,E)$ with

- edge weights $w(u,v)$
- source $s \in V$, sink $t \in V$

Cut of flow network G …

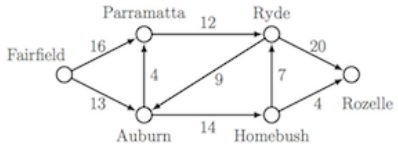
- a partition of V into $S \cup T$
  - $s \in S$, $t \in T$, S and T disjoint
- its *weight* is the sum of the weights of the edges between S and T:

$$\omega(S, T) = \sum_{u \in S} \sum_{v \in T} w(u, v)$$

Minimum cut problem … find cut of a network with minimal weight

---

Exercise #16: Cut of Flow Networks

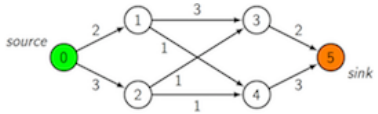


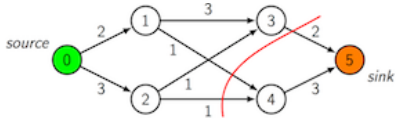
What is the weight of the cut {Fairfield,Parramatta,Auburn}, {Ryde,Homebush,Rozelle}?

12+14 = 26

Exercise #17: Cut of Flow Networks

Find a minimal cut in:





$\omega(S,T) = 4$

... Sidetrack: Maxflow and Mincut

Max-flow Min-cut Theorem.
In a flow network G the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G relative to f contains no augmenting path
3. value of flow f = weight of some minimum cut (S,T) of G

Randomised Algorithms for NP-Problems

Many NP-problems can be tackled by randomised algorithms that

- compute nearly optimal solutions
  - with high probability

Examples:

- travelling salesman
- constraint satisfaction problems, satisfiability
- … and many more

Simulation

Simulation

In some problem scenarios

- it is difficult to devise an analytical solution
- so build a software *model* and run *experiments*

Examples: weather forecasting, traffic flow, queueing, games

Such systems typically require random number generation

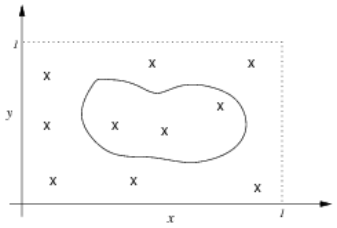
- distributions: uniform, numerical, normal, exponential

Accuracy of results depends on accuracy of model.

Example: Area inside a Curve

Scenario:

- have a closed curve defined by a complex function
- have a function to compute "X is inside/outside curve?"



... Example: Area inside a Curve

Simulation approach to determining the area:

- determine a region completely enclosing curve
- generate very many random points in this region
- for each point x, compute *inside(x)*
- count number of insides and outsides
- areaWithinCurve = totalArea * insides/(insides+outsides)

i.e. we approximate the area within the curve by using the ratio of points inside the curve against those outside

This general method of approximating is known as Monte Carlo estimation.

Summary

- Alphabets and words
- Pattern matching
  - Boyer-Moore, Knuth-Morris-Pratt
  - Tries
- Text compression
  - Huffman code
- Approximation
  - numerical problems
  - vertex cover
- Analysis of randomised algorithms
  - *probability of success*
  - *expected runtime*
- Randomised Quicksort
- Karger's algorithm
- Simulation

- Suggested reading:
  - tries … Sedgewick, Ch. 15.2
  - approximation … Moffat, Ch. 9.4
  - randomisation … Moffat, Ch. 9.3, 9.5

Algorithm and Data Ethics

Data Breaches

Major incidents …

- *TJ Maxx credit and debit card theft* (2005-07)

  Hackers gained access to accounts of over 100 million customers
  ⇒ Customers exposed to credit/debit card fraud

- *Yahoo! data breach* (2013-16)

  Hackers gained access to all 3 billion user accounts
  Details taken included names, DOBs, passwords, answers to security questions
  ⇒ Customers exposed to identity theft
  ⇒ Over 20 class-action lawsuits filed against Yahoo!

- *Facebook-Cambridge Analytica data scandal* (2018)

  Millions of people's Facebook profiles used for political purpose without their consent
  ⇒ Cambridge Analytica went bust as a consequence

... Data Breaches

The Guardian, 30/03/15 …

Personal details of world leaders accidentally revealed by G20 organisers

Exclusive: Obama, Putin, Merkel, Cameron, Modi and others kept in the dark after passport numbers and other details were disclosed in Australia's accidental privacy breach

- Follow our full coverage of this exclusive story
- Read the immigration department's letter outlining the circumstances of the G20 privacy breach

▲ Tony Abbott and Vladimir Putin cuddle koalas before the start of the first G20 meeting in November 2014. Photograph: Andrew Taylor/G20 Australia/Getty Images

The personal details of world leaders at the last G20 summit were accidentally disclosed by the Australian immigration department, which did not consider it necessary to inform those world leaders of the privacy breach.

The Guardian can reveal an employee of the agency inadvertently sent the passport numbers, visa details and other personal identifiers of all world leaders attending the summit to the organisers of the Asian Cup football tournament.

... Data Breaches

More severe, recent incidents in Australia …

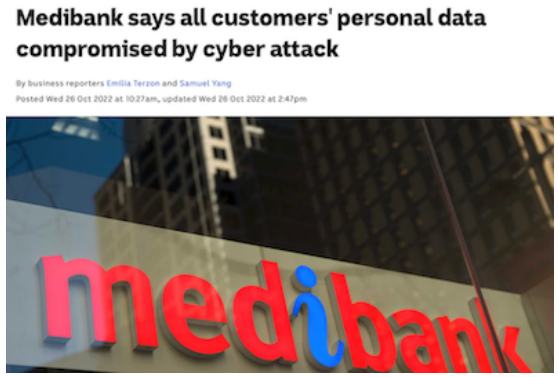
- *Optus cyberattack* (Sept 2022)

  Hacker gained access to personal information of 2.1 million costumers

  Details taken included names, DOBs, street addresses, driving licence numbers, passport numbers
  ⇒ Customers vulnerable to financial crimes
  ⇒ Up to 100,000 new passports had to be issued
  ⇒ Optus put aside $140 million for costs related to the breach

... Data Breaches

- *Medibank cyberattack* (Oct 2022)



Details of 9 million customers taken, including names, DOBs, street addresses, medical diagnoses and procedures

Also passport numbers and visa details for international students stolen

Ransom demanded but refused
⇒ Medical records posted on darknet, including data on abortions, mental health information

---

... Data Breaches

Australia's *Privacy Act 1988* …

- outlines how personal information must be used and managed
- applies to government agencies, businesses and organisations with annual turnover of >$3 million, private health services, …

Individuals have the right to:

- have access to their personal information
- know why and how information is collected and who it will be disclosed to
- ask to stop unwanted direct marketing

Businesses and organisations must comply with the *Australian Privacy Principles*:

- how to collect personal information
- how (not) to use personal information
- how to secure personal information

... Data Breaches

Australia's Privacy Act 1988 *Notifiable Data Breaches scheme*

In the event of a suspected or known data breach …

- contain breach where possible
- assess if personal information is likely to result in serious harm to affected individuals
  - individuals must be notified promptly
  - Australian Information Commissioner must also be notified
- take action to prevent future breaches

Data (Mis-)use

In 2012 several newspapers reported that …

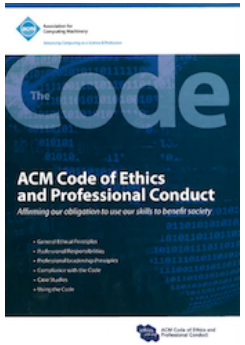
- Target used data analysis to predict whether female customers are likely pregnant
- Target then sent coupons by mail
- A Minneapolis man thus found out about the pregnancy of his teenage daughter

Not based on a factual story, but not implausible either

Who "owns" your data?

- big companies (Google, Meta, Microsoft, …)?
- governments?
- you?

... Data (Mis-)use



- Respect privacy
  - Store only the minimum amount of personal information necessary
  - Prevent re-identification of anonymised data
- Carefully analyse the consequences of data aggregation
- Access data only when authorised or compelled by the public good
  - Whistleblower Manning's disclosing of classified military data to Wikileaks (2010-11)
  - Paradise papers that disclosed offshore investments (2017)

Source: *ACM Code of Ethics and Professional Conduct*

Costly Software Errors

NASA's Mars Climate Orbiter …

- launched 11/12/1998
- reached Mars on 23/9/1999
- came too close to surface and disintegrated

Cause of failure:

- spec said impulse must be calculated in *newton seconds*
- one module calculated impulse in *pound-force seconds*
- 1 newton ≅ 0.2248 pound-force

... Costly Software Errors

Toyota vehicle recall (2009-11)

- Vehicles experienced sudden unintended acceleration
- 89 deaths have been linked to the failure
- 9 million cars recalled worldwide

Causes of failure included …

- a deficiency in the electronic throttle control system:
  stack overflow
  ⇒ stack grew out of boundary, overwrote other data

... Costly Software Errors

Sydney Morning Herald, 05/01/2010:

BUSINESS

Welcome to 2016: Eftpos glitch spreads

By Chris Zappone
5 January 2010 — 10:32am

The computer bug that brought Bank of Queensland's Eftpos transactions to a grinding halt in the first days of the New Year has hit other banks - including the Commonwealth Bank-owned BankWest.

The glitch, which started on January 1, left Australian retailers struggling to perform routine electronic point-of-sale transactions.

Merchants instead had to rely on carbon vouchers provided by banks or temporary measures that overrode the machines' internal time stamp.

Because of the error, Eftpos terminals recognise the year as 2016.

EFTPOS terminals inoperable for several days in early 2010

- customers' cards rejected as expired

Cause of failure:

- one module interpreted the current year as hexadecimal
  - `0x09 = 09`
  - `0x10 = 16 (≠ 10)`

Sidetrack: Year 2038 Problem

Recall:

```
#include <time.h>
time(NULL) // returns the time as the number of seconds
           // since the Epoch, 1970-01-01 00:00:00 +0000
```

Year 2038 problem …

- `time(NULL)` on 19 January 2038 at 03:14:07 (UTC) will be `2147483647 = 0x7FFFFFFF`
- a second later it will be `0x80000000` = -2,147,483,648
- ⇒ -2^{31} seconds since 01/01/1970 ("Epoch") is 13 December 1901 …

Programming Ethics

From the *ACM/IEEE Software Engineering Code* …

- Software engineers shall ensure that their products meet the highest professional standards possible
  - Strive to fully understand the specifications for software
  - Ensure that specifications have been well documented and satisfy the users' requirements
  - Ensure adequate testing, debugging, and review of software and related documents
- Approve software only if it
  - is safe
  - meets specifications
  - passes appropriate tests
  - does not diminish quality of life, diminish privacy or harm the environment

---

... Programming Ethics

Algorithms can save lives.

Uberlingen airplane collision 1/7/2002 at 11:35pm …

- passenger jet V9 2937 and cargo jet QY 611 on collision course at 36,000 feet
- ground air traffic controller instructed V9 pilot to descend
- seconds later, the automatic Traffic Collision Avoidance System (TCAS)
  - instructed V9 2937 to climb
  - instructed QY 611 to descend
- flight 611's pilot followed TCAS, flight 2937's pilot ignored TCAS
- all 71 people on board the two planes killed

⇒ Collision would not have occurred had both pilots followed TCAS

Exercise #18: Collision Avoidance Algorithm

The TCAS …

- builds 3D map of aircraft in the airspace
- determines if collision threat occurs
- automatically negotiates mutual avoidance manoeuvre
- gives synthesised voice instructions to pilots ("climb, climb")

What algorithm would you use for reaching an agreement (climb vs. descent)?

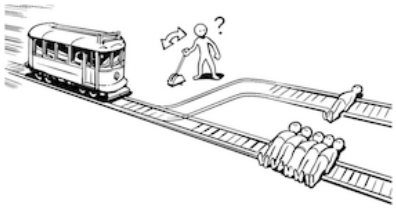
Moral Dilemmas

How to program an autonomous car …

- for a potential crash scenario
- when you have to choose between two actions that are both harmful

This is a modern version of the *Trolley Problem* …

- A runaway trolley is on course to kill five people
- You stand next to a lever that controls a switch
- If the trolley is diverted, it will kill one person on the side track



Is it ethical to pull the lever and kill the one in order to save the five?

Exercise #19: Moral Dilemmas

What would you do?

Variations:

- Fat man on bridge
- Transplant

⇒ try it yourself on the Moral Machine

Course Review

Course Review

Goal:

For you to become competent Computer Scientists able to:

- choose/develop effective data structures
- choose/develop algorithms on these data structures
- analyse performance characteristics of algorithms (time/space complexity)
- package a set of data structures+algorithms as an abstract data type
- represent data structures and implement algorithms in C

Assessment Summary


```
labwork = mark for programs/quizzes (out of 8+8)
midterm = mark for mid-term test    (out of 12)
assn    = mark for large assignment (out of 12)
exam    = mark for final exam       (out of 60)

if (exam >= 25)
   total = labwork + midterm + assn + exam
else
   total = exam * (100/60)
```

To pass the course, you must achieve:

- at least 50/100 for `total`

which implies that you must achieve at least 25/60 for `exam`

... Assessment Summary

Check your results using

```
prompt$ 9024 classrun -sturec
```

```
ClassKey: 25T0COMP9024
...
Exam: 43/60
assn: 8/12
labwork: 11.5/16
midterm: 9/12
total: 72/100
```

```
ClassKey: 25T0COMP9024
...
Exam: 23/60
assn: 8/12
labwork: 11.5/16
midterm: 9/12
total: 38/100
```

Final Exam

Goal: to check whether you have become a competent Computer Scientist

Requires you to demonstrate:

- understanding of fundamental data structures and algorithms
- ability to analyse time complexity of algorithms
- ability to develop algorithms from specifications

Lectures, problem sets and assignments have built you up to this point.

... Final Exam

2-hour exam on **Monday, 10 February**

CSE Computer Labs, your Time/Lab/Seat emailed to you

2 hours, reading time starts 10 minutes before beginning of exam, be there early

- 7 multiple-choice questions, 4 open questions
- Covers *all* of the contents of this course
- Each multiple-choice question is worth 4 marks (7 × 4 = 28)
  Each open question is worth 8 marks (4 × 8 = 32)
- Closed book, but you can bring one A4-sized sheet of your own *handwritten* notes (not typed, not photocopied)
- Bring *student ID card, your zPass, ballpoint pens, your A4-sheet*

... Final Exam

Sample prac exam available on Moodle

- 4 multiple-choice questions, 2 open questions
- maximum time: 60 minutes

- *sample* solutions provided upon completion

... Final Exam

Of course, assessment isn't a "one-way street" …

- I get to assess you in the final exam
- you get to assess me in UNSW's MyExperience Evaluation
  - go to *https://myexperience.unsw.edu.au/*
  - login using *zID@ad.unsw.edu.au* and your zPass

  Response rate (as at Friday last week): 29.2% 😰

  Please fill it out …

  - give me some feedback on how you might like the course to run in the future
  - even if that is "Exactly the same. I liked how it was run."

Revision Strategy

- Re-read lecture slides and example programs
- Read the corresponding chapters in the recommended textbooks
- *Review/solve problem sets*
- Attempt prac exam questions on Moodle
- Invent your own variations of the weekly exercises (problem solving is a *skill* that improves with practice)

Supplementary Exam

If you attend an exam

- you are making a statement that you are "fit and healthy enough"
- it is your only chance to pass (i.e. no second chances)

Supplementary exam *only* available to students who

- do *not* attend the final exam *and*
- apply formally for special consideration
  - with a documented and accepted reason for not attending

Assessment

Assessment is about determining how well *you* understand the syllabus of this course.

If you can't *demonstrate your understanding*, you don't pass.

In particular, we don't pass people just because …

- please, please, … my parents will be ashamed of me
- please, please, … I tried *really hard* in this course
- please, please, … I'll be excluded if I fail COMP9024
- please, please, … this is my final course to graduate
- etc. etc. etc.

Failure is a fact of life. For example, my scientific papers or project proposals get rejected sometimes too.

Summing Up …

So What Was the Real Point?

The aim was for you to become a better computer scientist

- more confident in your own ability to design data structures and algorithms
- with an expanded set of fundamental structures and algorithms to draw on
- able to analyse and justify your choices
- ultimately, enjoying the software design and development process

---

Finally …

Book 9
Epilogue

Thus spake the Master Programmer:

"Time for you to leave."

... Finally …

That's All Folks

&&

Good Luck with the Exam

and with your future studies

😀 👍 🎓

Produced: 31 Jan 2025