8. Algorithmic Methods

Frank Stephan

March 13, 2014

Al Khawarizmi

In the ninth century Muhammad ibn Musa al-Khawarizmi, a Persian mathematician, wrote a book, " Compendious Book on Calculation by Completion and Balancing", on Indian numerals and the decimal positional number system, in which he presented methods for adding, multiplying, dividing numbers, extracting square roots and calculating digits of π.

Assessing Algorithms

```
function productsum(ar)
{ var i; var j; var s=0;
  for (i in ar)
  { for (j in ar)
    { s = s+ar[i]*ar[j]; } }
  return(s); }
```

Let n be the number of array elements of ar.
Time complexity: $O(n^2)$;
Space complexity: $O(1)$.
Space for input does not count towards space complexity.

Algorithmic Methods

- Brute Force algorithms
- Greedy algorithms
- Divide and conquer algorithms
- Dynamic programming algorithms

Brute Force Algorithms

Brute force algorithms, also known as naïve algorithms, generate and test algorithms and exhaustive search algorithms, explore the entire search space.

Linear Search

Linear search seeks an element in list. It searches the list until it finds the element. In the worst case it searches the entire list. It is $O(n)$.

$$[1, 33, 24, 54, 67, 54, 32, 45, 6, 7, 8, 5, 6, 4, 3, 2]$$

```
1 function search(n, A)
2 { var i=-1;
3   do {i++;}
4   while (A[i] !=n && i < A.length)
5   return i; }
```

"code/linear_search.js"

Load the HTML into a browser

Powers

A brute force algorithm for computing the nth power of a number multiplies this number by itself n times. It is $O(n)$.

$$2^5 = 2 \times 2 \times 2 \times 2 \times 2$$

```
1 function power(n, m)
2 { var r=1;
3   for (i=0;i<m; i++) {r = r*n; M[0]=M[0]+1;}
4   return r;}
```

"code/power.js"

Load the HTML into a browser — The variable M[0] countes the number of multiplications.

Marriages

We have a list of n boys and a list of n girls. We want to print all possible marriages. The algorithm is $\Theta(n^2)$

```
1 function marriage(B, G)
2 {var i, j,t,s; t = "";
3 for (i=0; i < B.length; i++)
4       {for (j=0; j < G.length; j++)
5             {t = t+A[i] + " can marry " + B[j]+".<br>";
                  s++}}
6 return (s+" possible marriages.<br>"+t);}
```

Fibonacci

The Fibonacci function computes the nth Fibonacci number.

$$fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$$

Naïve Fibonacci

Recall the naïve Fibonacci algorithm we implemented in JavaScript.

```
1 function fibonacci(n)
2 { var f;
3   if (n == 0){f = 0;}
4   else if (n == 1){f = 1;}
5   else {f = fibonacci(n - 1) + fibonacci(n - 2);}
6   return f; }
```

"code/fibonacci1.js"

Load the HTML into a browser

Fibonacci

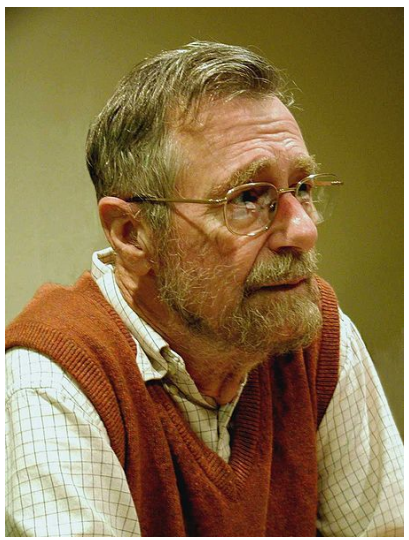
The algorithm computes the same values many times.

$$T(n) = T(n-1) + T(n-2) + \theta(1) = \varphi^n$$

φ is the Golden Ratio $= \frac{1+\sqrt{5}}{2} = 1.61803398875\cdots$

Shortest Path Problems

A shortest path problem could seek a shortest path between a single source and a single sink, all shortest paths between a single source and all vertices, or all shortest paths. The weights could be restricted to be positive, integers or could be bounded. Other constraints such as a maximum number of hops or a maximum total weight could be added. We study the single source, all sinks shortest path problem.



Brute Force Algorithm

1. Initialize the shortest path distance from the source s to itself to be $d(s) = 0$.
2. Initialize the current computed shortest distance from the source to a vertex v to $d(v) = \infty$ for all other vertices v.
3. Initialize the predecessor of a vertex v in the shortest path $\pi(v) = nil$ for all vertices v.
4. Repeat the following until nothing changes.
   1. Select an edge (u, v).
   2. If $d(v) > d(u) + W((u, v))$, then $d(v) = d(u) + W((u, v))$ and $\pi(v) = u$.

Contained Force Algorithm

1. Initialize the shortest path distance from the source s to itself to be $d(s) = 0$.
2. Initialize the current computed shortest distance from the source to a vertex v to $d(v) = \infty$ for all other vertices v.
3. Initialize the predecessor of a vertex v in the shortest path $\pi(v) = nil$ for all vertices v.
4. mark s as visited.
5. Repeat the following until nothing changes.
    1. Select an edge (u, v) where u is visited.
    2. If $d(v) > d(u) + W((u, v))$, then $d(v) = d(u) + W((u, v))$ and $\pi(v) = u$ and mark v as visited.

Greedy Algorithms

Greedy algorithm rely on a proof (needed) that the best local decisions lead to the best global solution.

Finding Peaks

A peak in a list of numbers is a number whose predecessor and successor are both lower than or equal to that number. If the first element of the list is larger than or equal to the second element, then it is a pick. If the last element of the list is larger than or equal to the one before last, then is a peak.

Peak Finding

The peak finding algorithm always moves up the slope. In the worst case it searches the entire list. It is $O(n)$.

```
1 function ascend(A,i)
2 { if (((i-1) in A) && (A[i-1] > A[i])) { return(i-1); }
3   if (((i+1) in A) && (A[i+1] > A[i])) { return(i+1); }
4   return(i); }
5 function findapeak(A)
6 { var i=0;
7   while (ascend(A,i) != i) { i = ascend(A,i); }
8   return(i); }
```

"code/peak.js"

Load the HTML into a browser

Too Greedy Shortest Path

Always take the next shortest edge. Does it work?

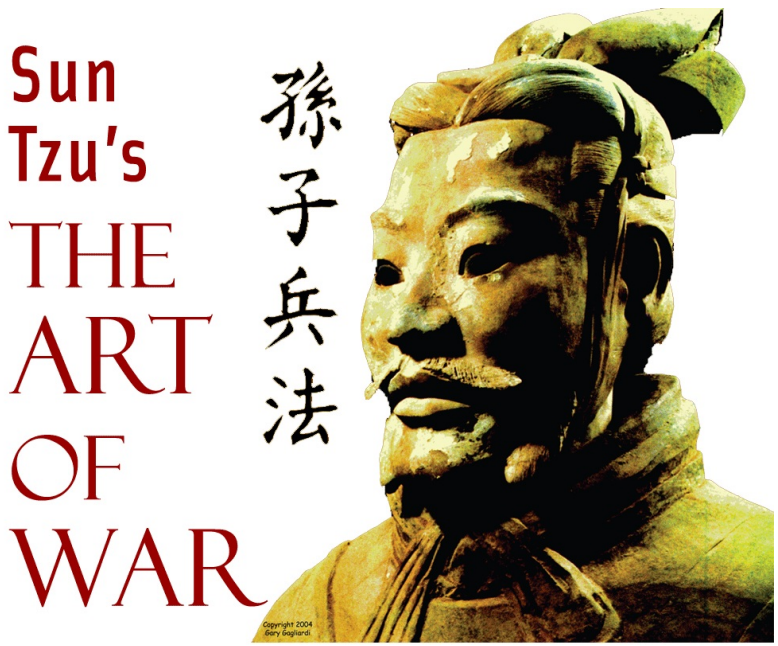
Optimal Substructure

The solutions to local problems must contribute to a global solution.

For example, the function findapeak finds a local peak but not necessary the global peak; it might settle with an intermediate peak or just an even plain.

Divide and Conquer

Divide and conquer algorithms decompose the problem into smaller sub-problems and combines the results. This naturally leads to recursive algorithms.



$$n$$
$$|$$
$$\frac{n}{2}$$
$$|$$
$$\cdots$$
$$|$$
$$O(1)$$

$$O(\log(n))$$

Dichotomic Peak Search

Dichotomic peak iteratively halves the search interval from i to j and searches in that half to which the slope in the middle points. It is $O(\log n)$.

```
1  function ascend(A,i)
2  { if (((i-1) in A) && (A[i-1] > A[i])) { return(i-1); }
3    if (((i+1) in A) && (A[i+1] > A[i])) { return(i+1); }
4    return(i); }
5  function findapeakfast(A)
6  { var i=0; var j = A.length-1;
7    if (ascend(A,i) == i) { return(i); }
8    if (ascend(A,j) == j) { return(j); }
9    var k = Math.round((i+j)/2);
10   while (ascend(A,k) != k)
11     { if (ascend(A,k) < k) { j = k; } else { i = k; }
12       k = Math.round((i+j)/2); }
13   return(k); }
```

"code/peakfast.js"

Binary Search

Binary seeks an element in a sorted list. It recursively searches one half of the list. In the worst case it has to divide the list until it contains only one element. It is $O(\log n)$.

$$T(n) = T(\frac{n}{2}) + \Theta(1)$$

Master Theorem

Let $a \geq 1$ and $b > 1$ be constants. Let $f(n)$ be a function. Let $T(n)$ be a function on the non-negative integers by the following recurrence[a].

$$T(n) = a \times T(\frac{n}{b}) + f(n)$$

T(n) can be bounded asymptotically as follows.

1. If $f(n) \in O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b(a)})$.
2. If $f(n) \in \Theta(n^{\log_b(a)})$, then $T(n) \in \Theta(n^{\log_b(a)} \times log_b(n))$
3. If $f(n) \in \Omega(n^{\log_b(a)+\epsilon})$ for some constant $\epsilon > 0$, and if $a \times f(\frac{n}{b}) \geq c \times f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$.

[a] $\frac{n}{b}$ means either $\lceil \frac{n}{b} \rceil$ or $\lfloor \frac{n}{b} \rfloor$.

$$T(n) = T(\frac{n}{2}) + \Theta(1)$$

$$T(n) = \Theta(n^{\log_2(1)} \times log_2(n)) = \Theta(\log n)$$

```
1  function search(n, A)
2  {var r=bsearch(0, A.length-1, n, A);
3  return r;}
4
5  function bsearch(s, e, n, A)
6  {var r;
7  if (s <= e){
8    var pivot = Math.floor(s+ ((e - s) /2));
9    if (A[pivot] < n) {r= bsearch(pivot+1,e,n,A);}
10   else if (A[pivot] > n) {r = bsearch(s,pivot-1,n,A);}
11   else {r=pivot;}}
12 else {r=-1;}
13 return r;}
```

"code/binary_search.js"

Load the HTML into a browser

Binary Search for Unsorted Lists

Why not sorting a list before searching it?

- We can sort a list is $O(n \log n)$ and then search it in $O(\log n)$. This is $O(n \log n)$. This is still worse than $O(n)$.
- If we need to search the list multiple times (say k times). It is worth the effort if $k > \log(n)$.

Powers

A naive divide and conquer algorithm for computing the nth power repeatedly divides the exponent by two. It is $O(n)$.

$$2^5 = (2 \times 2) \times (2 \times 2 \times 2)$$

```
1 function power(n, m)
2 { if (m < 1) { return(1); }
3   if (m == 1) { return(n); }
4  M[0] = M[0]+1;
5   var k = Math.floor(m/2);
6   return(power(n,k)*power(n,m-k)); }
```

"code/power1.js"

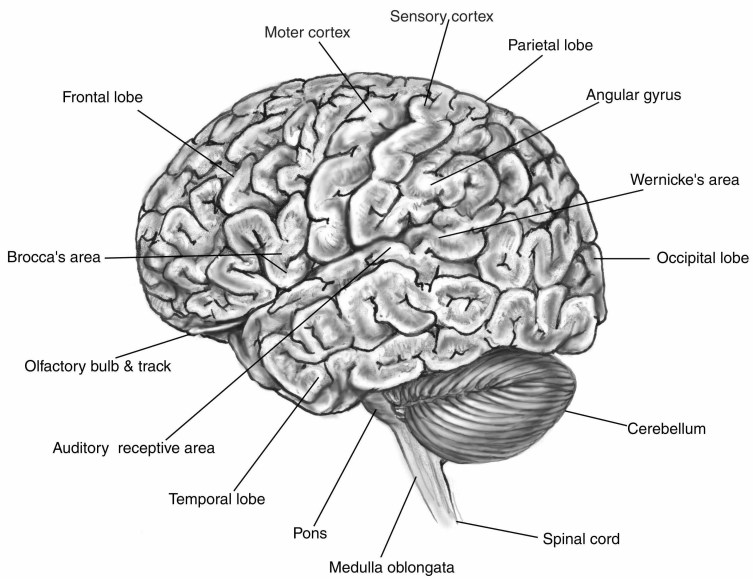
Load the HTML into a browser

Dynamic Programming

Dynamic programming was invented by Richard Bellman working for the RAND. The name "because it is something that not even a congressman could object to".

Dynamic Programming

Dynamic programming combines divide and conquer with memoization, namely recording and reusing the solutions to subproblems.



Powers

A dynamic programming algorithm for computing the nth power repeatedly divides the exponent by two and uses memoization. It is $O(\log n)$.

```
1 function power(n, m, R)
2 { if (R[m] == undefined)
3   { if (m < 2) { R[0] = 1; R[1] = n; }
4     else { var k = Math.floor(m/2);
5            R[k] = power(n,k,R);
6            R[m-k] = power(n,m-k,R);
7            R[m] = R[k]*R[m-k]; M[0]=M[0]+1; } }
8   return(R[m]); }
```

"code/power2.js"

Load the HTML into a browser

Squaring in a function

One can also avoid the multiple evaluation of the argument by having a square function which multiplies the value with itself without twice calculating it.

```
1 function square(n)
2 { M[0]++; return(n*n); }
3 function power(n, m)
4 { if (m==0) { return(1); }
5   if (m==1) { return(n); }
6   if (m%2 == 0)
7     { return(square(power(n,m/2))); }
8   if (m%2 == 1)
9     { M[0]++; return(n*square(power(n,(m-1)/2))); }}
```

"code/power3.js"

Load the HTML into a browser

Comparison

Let us compare the algorithms.

Load the HTML for brute force power into a browser power.js with input (1.001,1023) computes 21916.681339054456 with 10000 multiplications.
Load the HTML for divide and conquer power into a browser power1.js with input (1.001,10000) computes 21916.681339042156 with 9999 multiplications.
Load the HTML for dynamic programming power with memoization into a browser power2.js with input (1.001,10000) computes 21916.681339042156 with 21 multiplications.
Load the HTML for dynamic programming power with direct reuse into a browser power3.js with input (1.001,10000) computes 21916.681339053735 with 17 multiplications.

Fibonacci

Now we only recurse the first time we compute the k^{th} Fibonacci number. for the rest we read the value. The algorithm is now $\Theta(n)$.

```
1 function fibonacci(n,F)
2 { if (F[n]== undefined) // if calculations needed
3   { if (n < 2) { F[n] = n; } // Cases n==0 and n==1;
4     else { F[n] = fibonacci(n-1,F)+fibonacci(n-2,F); }}
5   return(F[n]); } // F[n] is fibonacci(n)
```

"code/fibonacci2.js"

Load the HTML into a browser

Fibonacci

Every recursive program can be transformed into an iterative program. The iterative Fibonacci function computes the values bottom-up instead of top-down.

```
1 function fibonacci(n)
2 { var i;
3   var F = new Array(0,1,1); // Base Case
4   for (i=3;i<=n;i++) // Inductive Definition
5   { F[i] = F[i - 1] + F[i - 2]; }
6   return F[n]; }
```

"code/fibonacci3.js"

Load the HTML into a browser

Fibonacci

The iterative Fibonacci function can be computed in constant space. It only requires remembering the last three values.

```
1 function fibonacci(n)
2 { var i; var F = new Array(0,1,1);
3   for (i=0;i<n;i++)
4   { F[0] = F[1]; F[1] = F[2]; F[2] = F[0]+F[1]; }
5   return F[0]; }
```

"code/fibonacci4.js"

Load the HTML into a browser

Dijkstra Algorithm

1. Initialize the shortest path distance from the source s to itself to be $d(s) = 0$.
2. Initialize the current computed shortest distance from the source to a vertex v to $d(v) = \infty$ for all other vertices v.
3. Mark all vertices as unvisted.
4. While there are unvisted vertices do
   1. Select among the unvisited vertices the current vertex c such that $d(c)$ is as small as possible.
   2. For every edge of the form (c, v) do
      1. If $d(v) > d(c) + W((c, v))$, then $d(v) = d(c) + W((c, v))$.
   3. Mark the current vertex c as visited.

Project on a Computer Game

The task is to implement the game of Hexa Reversi and to make a good strategy for the computer to play the game.

Read the Project Description with your browser

Presentation: Thursday 17 April 2014 in Lecture

Deadline for all material: Wednesday 16 April 2014 24:00 hrs

Send email to Thomas Kister (kister@comp.nus.edu.sg) with name of team members and urls of material and game and player

Overview

The project has the goal to write the environment for an interactive game of Hexagonal Reversi (Hexareversi for short).

- Players play on a hexagonal board having 61 fields.
- Each field has six neighbours.
- The players move alternately.
- A player can put a new piece iff it captures pieces of the opponent between the new position and some old piece of the player; all the captured pieces are turned into pieces of the player.
- A player can also pass; if there is no possible move, a player has to pass.
- When both players pass one after the other, the game ends. The player with the most pieces wins.

Player X puts new piece "x"

```
          _   _   _   _   _
        _   _   _   _   _   _
      _   _   X   X   X   Y   _
    _   _   X   X   X   y   _   _
  _   _   Y   Y   Y   Y   y   _   _
    _   _   Y   X   Y   Y   y   _
      _   _   X   X   y   y   x
        _   _   _   _   _   _
          _   _   _   _   _
```

When putting the piece "x" the player will capture five pieces "y" (all new and changed pieces in lower case).

Situations and Scoring

In the following, some general ideas on how to implement strategies are given. These strategies use the following assumptions:

- A situation in the game consists of the current board, the number of passes just before (0–2) and the player who has to move (X or Y).
- The starting situation has 10 random pieces on the board and player X is to move and no passes before the current.
- Every situation in the game has a score which is Number(X Pieces) - Number(Y Pieces).
- When both players pass or have to pass, the game ends and the score determines the winner: positive number means Player X wins; negative number means Player Y wins; zero score means Draw.

Sample Situation

```
            X   X   X   X   X
          X   X   X   X   X   X
        X   X   X   X   X   X   _
      X   X   X   X   X   X   _   _
    Y   Y   Y   Y   Y   Y   Y   _   _
      Y   Y   Y   Y   Y   Y   Y   _
        Y   Y   Y   Y   Y   Y   Y
          X   X   Y   Y   Y   Y
            X   X   Y   Y   Y
```

`Number of Passes: 2; Player to Move: X`

In this situation, both players passed (had to pass) and the game ended. X has 27 pieces and Y has 28 pieces, so the score is -1 and Y wins.

Possible Moves

```
            X   X   X   X   X
          X   X   X   X   X   Y
        X   X   X   X   Y   Y   1
      X   X   X   X   Y   Y   2   _
    Y   Y   Y   Y   Y   Y   Y   _   _
      Y   Y   Y   Y   Y   Y   Y   _
        Y   Y   Y   Y   Y   Y   Y
          X   X   Y   Y   Y   Y
            X   X   Y   Y   Y
Number of Passes: 1; Player to Move: X
```

X can move to position 1 and score three pieces; Y will have to pass.
X can move to position 2 and score three pieces; Y can retake one piece.

Score of a Situation

Calculating Scores

- Let S be the set of possible situations and $s \in S$.
- If the game has ended in situation s then $score(s)$ is the difference Number(Pieces of X) - Number(Pieces of Y).
- If Player X has to move in situation s then $score(s) = \max\{score(t) : \text{X can reach } t \text{ from } s \text{ by making one move or passing}\}$.
- If Player Y has to move in situation s then $score(s) = \min\{score(t) : \text{Y can reach } t \text{ from } s \text{ by making one move or passing}\}$.

Strategy: Each player chooses the move or passes such that the score is preserved. If there are several options giving the same score, any of these moves is as good as the others.
Problem: This is too complicated for being computed.

Ways Out

- Just assume that the target of the move is a final situation and take the one with the highest score (not that good);
- Similar as before, but do a weighted random choice (better situations have higher odds, Greedy Random Player);
- Simulate the game for all possible moves for c rounds and then evaluate the situations (assuming that they are final), compute back the scores of the situations on the way to them and take the best possible move by these estimated scores (game tree method);
- As before, but ignore obviously bad moves (pruned game tree method);
- Evaluate situations by different methods than the score (Position Random Player does this, but it does not simulate the game tree).

Value of Positions

			9		3		6		3		9			
				3		1		2		1		3		
			6		2		3		3		2		6	
		3		2		3		3		3		2		3
	9		1		3		3		3		3		1		9
		3		2		3		3		3		2		3
			6		2		3		3		2		6	
				3		1		2		1		3		
					9		3		6		3		9	

Possible quality of positions to move: Corners are extremely good, because they are never lost; neighbouring positions are not that good, as going there might give away the corner. The Position Random Player uses this hierarchy of position qualities - nevertheless, as it does not search the game tree, a human can easily defeat it.

Summary of Project

- Read instructions on `http://www.comp.nus.edu.sg/~gem1501/year1314sem2/project/index.html`.
- Register your team with an email to kister@comp.nus.edu.sg and select a team id from "t00" through "t95". Name all team members in the email (ideally should be two).
- Write a completely new project from scratch or improve one of the available prototypes.
- Make a better user interface (move by clicking on fields in place of typing field numbers) and incorporating tournament options for comparing strategies and making an own strategy in a separate js-file.
- Stick to the programming conventions and instructions outlined in the project description.

Summary of Project - Continued

- Put your project including some explanations on what you did and how your project works and presentation-slides onto your homepage.
- Make a presentation in the last lecture for 5 minutes on 17 April 2014.
- All material should be ready by 16 April 2014 24:00 hrs and let us know the url by email to kister@comp.nus.edu.sg.
- Available prototypes: More advanced: `http://www.comp.nus.edu.sg/~gem1501/year1314sem2/project/hexareversi.html` with an included js-file and some more basic and incomplete version is on `http://www.comp.nus.edu.sg/~fstephan/hexareversi.html`.

Attribution