

GEM 1501 Problem Solving With Computers

Lecture 4:

Algorithmic Methods

Frank Stephan

Summary of Previous Lecture

- Syntax and Semantics of Programming Languages
- Compilers and Interpreters
- Collection of Programming Languages
- Data Types in Java Script

Various Programming Languages

- No single solution for everything, languages evolve in time, laws of competition and market
- Classical Languages: Fortran, Pascal, Basic, C
- Specialized Languages:
 - APL (vector and tensor calculations)
 - Snobol (text processing)
 - Prolog (solving logic problems)
 - Java Script (programs embedded into webpages)

Structured Programming

Current: Structured Programming

```
/* Example: Java Script Function */  
  
function factorial(n)  
  { var m=1; var k;  
    for (k=n;k>1;k=k-1)  
      { m = m*k; }  
    return(m); }  
  
var n = window.prompt("Input n");  
window.alert(n+"! is "+factorial(n));
```

Old: Line number + Goto

```
10 rem Basic Loop  
20 input n  
30 k = n  
40 m = 1  
50 if k < 1 goto 90  
60 m = m*k  
70 k = k-1  
80 if k > 1 goto 60  
90 print "Factorial ";  
100 print "of ";n;" is ";m
```

Java Script Objects

- Array: Collection of many similar items:

```
ar == (ar[0],ar[1],ar[2],ar[3],ar[4],ar[5],ar[6])
```

```
ar = new Array(1,1,2,3,5,8,13); or
```

```
ar = new Array(7); ar[0] = 1; ar[1] = 1; ... ar[6] = 13;
```

- Record: Collection of few different items:

```
re == (re.sname,re.gname,re.num)
```

```
function remake(a,b,c)
```

```
{ this.sname = a; this.gname = b; this.num = c; }
```

```
re = new remake("Aal","Anneliese",1);
```

- <http://www.comp.nus.edu.sg/~gem1501/salary.html>

Overview of Today's Lecture

- **Searches and traversals**
- Divide and conquer
- Greedy algorithms
- Dynamic programming

Searches and Traversals

- Data structures provide complex information
- Structures have information implicit
- Searches and traversals make information explicit

Search in Books

- Find something on “Algorithmic Methods” in textbook of this lecture.
 - Exhaustive Search: Read book from beginning to end.
 - Contents: Read table of contents completely and then Chapter 4 on pages 81–98.
 - Index: Read starting on page 495 until keyword “Algorithmic Methods” and then on the referred pages 81–98.
- Avoiding Exhaustive Search needs additional information.
 - Either some properties of problem can be exploited
 - Or additional sources of information exist (as table of contents or indices in books)

Search in the Internet

- Links from pages to other: Course Home page has links to slides, homework, Introduction to Java Script and so on.
- Search machines: read all available pages regularly; maintain indices for possible keywords; rank pages by importance.

- Searching “GEM 1501” in Google: 15 entries. First two:

GEM 1501 Home Page

A short introduction to Java Script for GEM1501 is available. This introduction is of course not as perfect as a text book but it should contain the most ...

www.comp.nus.edu.sg/~gem1501/index.html - 18k

GEM 1501 Home Page for year 2004

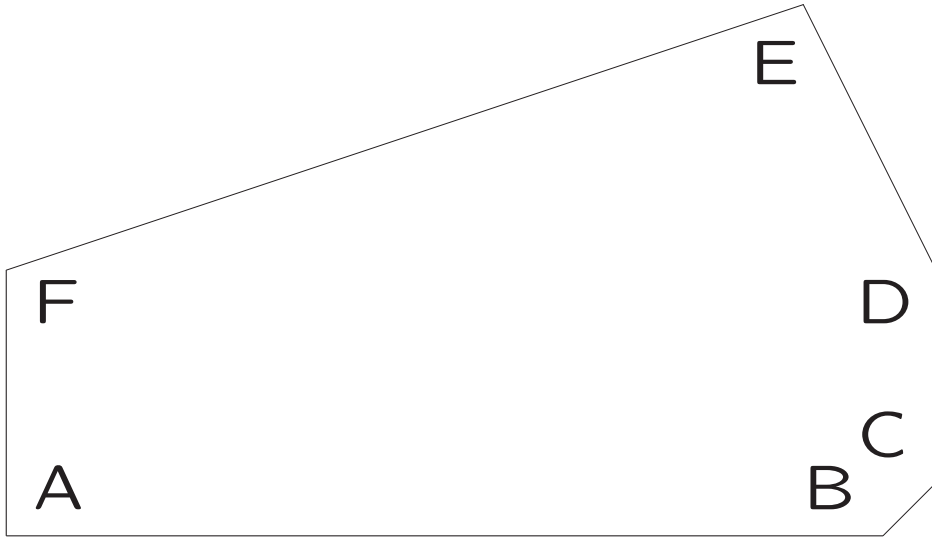
GEM 1501 - Problem Solving For Computing. Brief Description; Covered Topics; Material; Assignments and Assessments; Other Course Information ...

www.comp.nus.edu.sg/~gem1501/year2004.html - 13k

Search in Mathematics

- Find places where function is 0
- Find places where a function is maximal
- Find solutions to systems of equations
- Find functions which solve given differential equation

Maximal polygonal distance



What is longest distance of two points in convex polygon?

$$\text{Distance } AD = \sqrt{350^2 + 100^2} = 364.00549;$$

$$\text{Distance } AE = \sqrt{300^2 + 200^2} = 360.55513;$$

$$\text{Distance } CF = \sqrt{350^2 + 080^2} = 359.02646;$$

$$\text{Distance } BF = \sqrt{330^2 + 100^2} = 344.81879.$$

Search Strategies

- Exhaustive Search

Compute for each line AB, AC, AD, AE, AF, BC, BD, BE, BF, CD, CE, CF, DE, DF, EF the length and then find the line of maximal length.

- These are $\frac{1}{2}n(n - 1)$ computations for an n -gon.

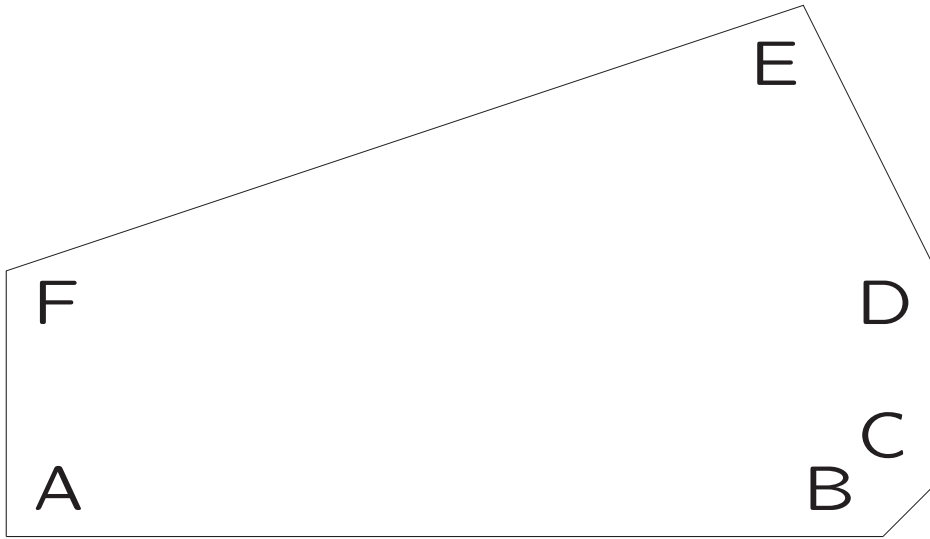
- Intelligent Search

For each side of the n -gon, rotate it such that the side is parallel to the bottom. Then find the one or two vertices with maximal height and compute its distance from the two bottom points of the n -gon. Take among all the line of maximal length.

- These are at most $4n$ computations for an n -gon but it needs rotating.

- Well-implemented, Intelligent Search is faster than Exhaustive Search although rotating might need time.

Rotating the Polygon



Bottom Lines: AB BC CD DE EF AF

Opposite Vertices: E F F,A A B C,D

AB, AF, BC, BD, CD, CE, DE, EF need not to be considered.

A Combinatorial Search-Problem

- (1) Four people and four colours;
- (2) Each person wears cloths of one colour;
- (3) Thomas wears red or green;
- (4) Angela dislikes blue and green;
- (5) Sarah likes blue and yellow;
- (6) Angela, Sarah and Jill wear different colours;
- (7) Thomas and Jill wear the same colour.

Is there a solution? If so, is it unique?

4 Solutions

Person	Colour 1	Colour 2	Colour 3	Colour 4
Thomas	red	green	green	green
Angela	yellow	red	red	yellow
Jill	red	green	green	green
Sarah	blue	yellow	blue	blue

Exhaustive Search until two solutions are found.

Takes long: n^m possibilities for n people and m colours.

No “really fast” algorithm known.

Overview of Today's Lecture

- Searches and traversals
- **Divide and conquer**
- Greedy algorithms
- Dynamic programming

Divide-and-Conquer

- If the problem is small enough, the solution is immediate
- If not, split the problem into smaller ones.
- Solve the smaller problems separately.
- Combine the solutions of the smaller problems into a solution of the bigger problem.

Examples for Divide-and-Conquer

- Towers of Hanoi
- Finding maximal element in a list
- Mergesort
- Pivotsort

The Towers of Hanoi

- Three Pegs A, B, C
- M Rings R_1, R_2, \dots, R_M
- Initially all rings on peg A
- At the end all rings should be on peg B
- Order: If $I < J$ then R_I can never be below R_J
- Moves: In each step, only one ring can be moved

Moving Towertops is Divide and Conquer

- $\text{Move}(1, X, Y, Z)$ is a small problem directly solvable which moves R_1 from X to Y .
- $\text{Move}(N, X, Y, Z)$ with $N > 1$ is a series of moves.
Solve Subproblem $\text{Move}(N - 1, X, Z, Y)$ recursively.
Move R_N from X to Y .
Solve Subproblem $\text{Move}(N - 1, Z, Y, X)$ recursively.
- Main Procedure is calling $\text{Move}(M, A, B, C)$.

Finding Maximal Element of List

- Input: Unsorted list of integers L ;
- Output: Largest element of L ;
- Sample Input: (0, 2, 8, 5, 343, 112, 6, 13, 1888, 256, 207);
Sample Output: 1888.

Divide-and-Conquer Algorithm

- If L has one element, this element is the largest element MAX.
- If L has more elements, then split L in L_{left} and L_{right}
- Compute the maximal elements MAX_{left} and MAX_{right} of L_{left} and L_{right} , respectively.
- Then MAX is the maximum of MAX_{left} and MAX_{right} .

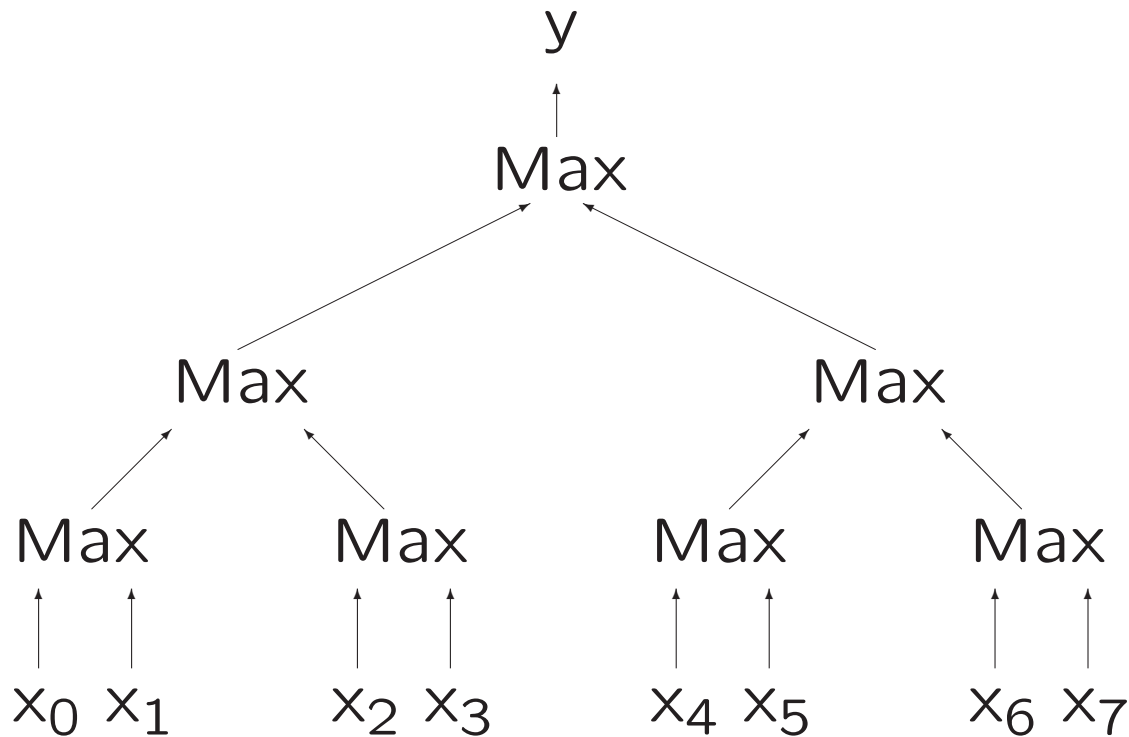
Maximum Search

```
function findmax(list)
  { var k; var r; var left; var right;
    k = Math.floor(list.length/2);
    if (k > 0)
      { left = list.slice(0,k);
        right = list.slice(k,list.length);
        r = Math.max(findmax(left),findmax(right)); }
    else
      { r=list[0]; }
    return(r); }

var fulllist = new Array(); ....
var mm = findmax(fulllist);
document.write("The maximum of the list is "+mm+"<br>");
```

<http://www.comp.nus.edu.sg/~gem1501/maximum.html>

Maximum as Circuit



Divide and Conquer implemented as a Circuit.

Each Maximum-Gate combines either two inputs or two subproblems below.

Mergesort

- Input: unsorted list of integers
- Output: sorted list with the same elements
- Idea:
 - If the list has one element, it is sorted.
 - If not, split it in half, sort the halves and merge them.

Runtime Example

```
input: (6,5,2,4,3,9,7,8)
split in two sublists: (6,5,2,4) | (3,9,7,8)
recursive call, split: (6,5) | (2,4) | (3,9) | (7,8)
recursive call, split: (6) | (5) | (2) | (4) | (3) | (9) | (7) | (8)
end of recursive calls: (6) | (5) | (2) | (4) | (3) | (9) | (7) | (8)
merge sublists: (5,6) | (2,4) | (3,9) | (7,8)
merge sublists: (2,4,5,6) | (3,7,8,9)
merge sublists, output: (2,3,4,5,6,7,8,9)
```

Merging two Sorted Lists

- Lists viewed upon as Queues
- Given: Sorted Lists $left, right$
Desired: Combined Sorted List $list$
- Initialize $list$ as empty
While $left$ and $right$ are both not empty
 - if $left[0] < right[0]$
 - then move $left[0]$ to the end of $list$
 - else move $right[0]$ to the end of $list$
- If exactly one of the lists $left, right$ is not empty then append the elements of it to $list$.
- Now $list$ is obtained by merging $left$ and $right$, done.

Runtime Example: Merging

Merge the lists (2,4,5,6) and (3,7,8,9)

left	right	list	
(2,4,5,6)	(3,7,8,9)	()	comparing and moving
(4,5,6)	(3,7,8,9)	(2)	first elements
(4,5,6)	(7,8,9)	(2,3)	
(5,6)	(7,8,9)	(2,3,4)	
(6)	(7,8,9)	(2,3,4,5)	
()	(7,8,9)	(2,3,4,5,6)	moving the remaining list
()	(8,9)	(2,3,4,5,6,7)	
()	(9)	(2,3,4,5,6,7,8)	
()	()	(2,3,4,5,6,7,8,9)	done

Mergesort in Java Script

```
function mergesort(list)
{
  var k = Math.floor(list.length/2);
  if (k<1) { return; }
  var left = new Array(); var right = new Array();
  while (list.length>k) { left.push(list.shift()); }
  while (list.length>0) { right.push(list.shift()); }
  mergesort(left); mergesort(right);
  while ((left.length > 0) && (right.length > 0))
    { if (left[0] < right[0]) { list.push(left.shift()); }
      else { list.push(right.shift()); } }
  while (left.length > 0) { list.push(left.shift()); }
  while (right.length > 0) { list.push(right.shift()); }
  return; }
}
```

<http://www.comp.nus.edu.sg/~gem1501/mergesort.html>

Pivot Search

- Input: unsorted list of integers
- Output: sorted list with the same elements
- Algorithm:
 - Move first element into variable "pivot".
 - Put other smaller elements into sublist "lower"
 - Put other larger elements into sublist "upper"
 - Sort those sublists which are not empty.
 - Result = lower + pivot + upper;
- <http://www.comp.nus.edu.sg/~gem1501/pivotsort.html>

Java Script Implementation

```
function pivotsort(list)
  { if (list.length < 2) { return; }
    var pivot = list.shift(); var current;
    var lower = new Array();
    var upper = new Array();
    while(list.length > 0)
      { current = list.shift();
        if (current < pivot) { lower.push(current); }
        else { upper.push(current); } }
    pivotsort(lower); pivotsort(upper);
    while (lower.length > 0) { list.push(lower.shift()); }
    list.push(pivot);
    while (upper.length > 0) { list.push(upper.shift()); }
    return; }
```

Runtime Examples

A: (3,2,5,8,4,6,1) -> pivot = 3, lower = (2,1), upper = (5,8,4,6)
-> B:pivotsort(2,1), C:pivotsort(5,8,4,6) -> (1,2,3,4,5,6,8)

B: (2,1) -> pivot = 2, lower = (1), upper = () -> (1,2)

C: (5,8,4,6) -> pivot = 5, lower = (4), upper = (8,6)
-> D:pivotsort(8,6) -> (4,5,6,8)

D: (8,6) -> pivot = 8, lower = (6), upper = () -> (6,8)

4 nontrivial calls of function pivotsort,
11 comparisons (6 in A, 1 in B, 3 in C, 1 in D).

L: (5,5,5,5,5) -> pivot == 5, lower == (), upper == (5,5,5,5)
5 iterations to process this sorted list.

This implementation is inefficient if same number fed in high multitude.

Overview of Today's Lecture

- Searches and traversals
- Divide and conquer
- **Greedy algorithms**
- Dynamic programming

Greedy Algorithms

- Idea: Start somewhere and construct the solution step-by-step
- Leads to very efficient algorithms
- Works for some problems, but by far not all
- Requires proof of correctness

Going to top of Mountain

Always take the way which goes up steepest
Is correct if there is one peak

```
      x
    xxxxxxx
  xxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx P
```

A problem if there are two peaks,
local and global behaviour might differ

```
      x
    xxxx          xxxxx
  xxxxxxxxxxxxxx  xxxxxxx
xxxxxxxxxxxxxxxxxxx xxxxxxxx
xxxxxxxxxxxxxxxxxxx P xxxxxxxx
```

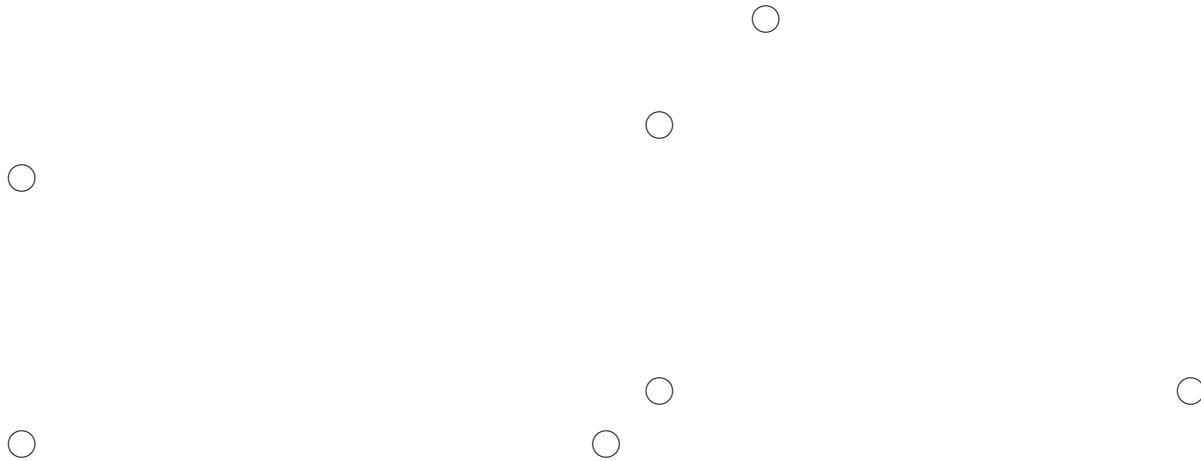
Examples

- Railroad contractor
- Navigation in a source/sink directed graph

Railroad Contractor

- Goal: Construct cheapest railroad system for a country
- Input: Set of cities and distances between them
- Output: Shortest network connecting all cities
- Problem is known as “Minimal Spanning Tree Problem”

The cities



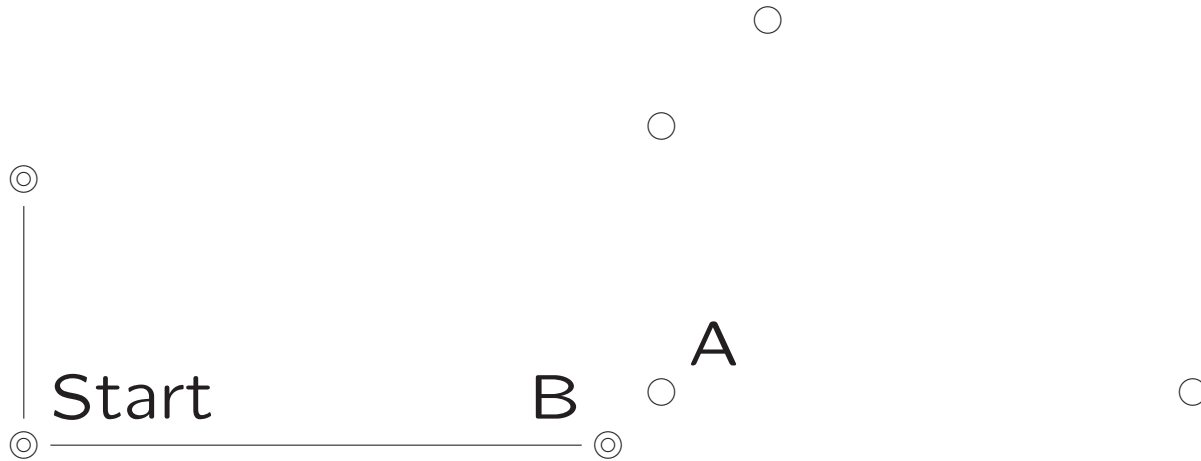
Railway Contractor wants to build small network of 6 lines

Meeting contract with as little afford as possible

Greedy Algorithm

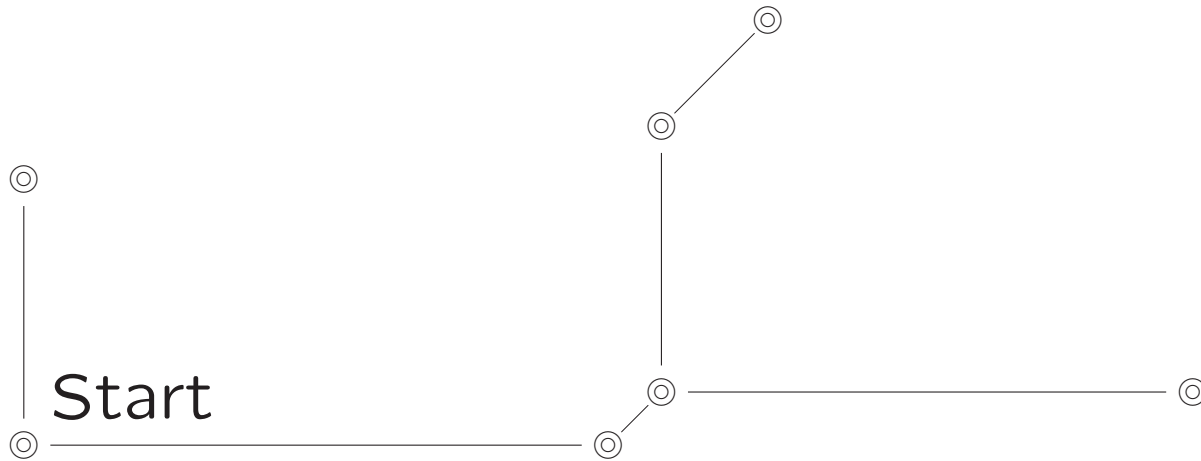
- Start with the network consisting of 1 city and 0 line;
- While the network does not cover all cities
{Search for city A nearest to the the current network;
Connect A to nearest city B within the network; }

Network of three cities



City A will be connected to City B in order to increase network.

Final Network

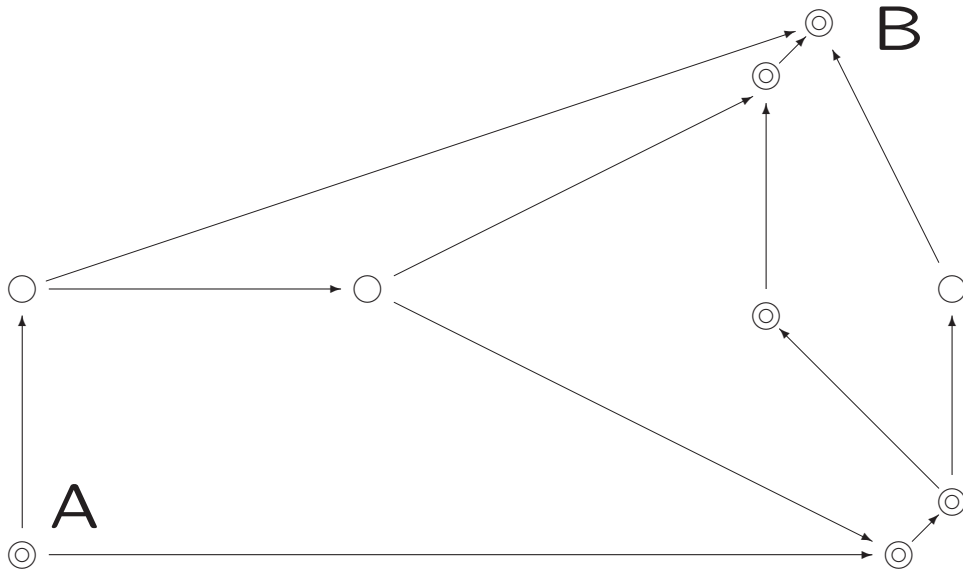


This network is also called “minimal spanning tree” of the given nodes.

Navigation in a Source/Sink Directed Graph

- Input: Directed acyclic labelled graph, where A is the only node that has no incoming edge and B is the only node that has no outgoing edge.
- Output: A path from A to B

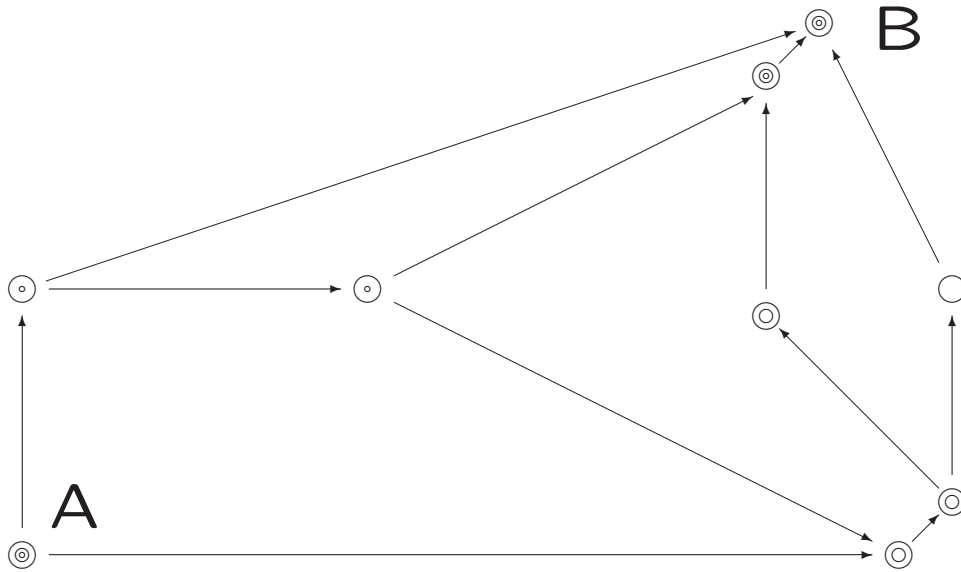
Example



Construct path by taking from each node the longest arrow and adding the arrow and target node to the path.

Algorithm gives a 5-hop-path from A to B.

As few hops as possible



Greedy algorithm with longest arrow: 5 hops

Greedy algorithm with shortest arrow: 4 hops

Other method needed -> Dynamic Programming

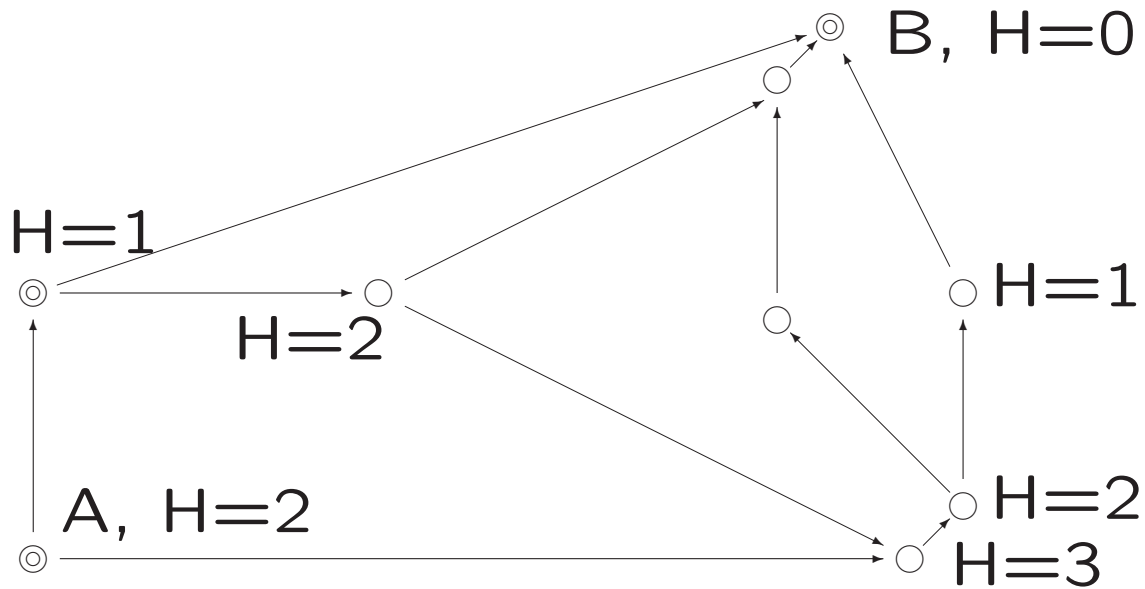
Overview of Today's Lecture

- Searches and traversals
- Divide and conquer
- Greedy algorithms
- **Dynamic programming**

Dynamic Programming

- Idea: On the path to a solution, accumulate useful information
- Re-use this information whenever needed
- Often surprisingly efficient algorithms
- Applied on a wide variety of problems

Minimizing Number of Hops



Define $H(B) = 0$ and for all X different from B let

$H(X) = \text{Min}\{H(Y)+1: \text{from } X \text{ is an arrow to } Y\}$.

When computation completed, go downward along the H values.

Price Minimization

- Input

Function $Price(X, Y) ==$ Price for directly going from X to Y

Output

Cheapest Voyage X_0, X_1, \dots, X_m with $0 = X_0 < X_1 < \dots < X_m = N - 1$;

- Compute Functions

$Next(X) == Y$ for going from X to Y , $Y > X$

$Costs(X)$ for price of going from X to B

- Algorithm

Let $Costs(N - 1) = 0$ and let $Next(N - 1) = N - 1$;

For $X = N - 2$ downto 0

{ Choose $Y > X$ such that $Costs(Y) + Price(X, Y)$ is minimal;

Let $Next(X) = Y$ and let $Costs(X) = Price(X, Y) + Costs(Y)$; }

- Voyage

From 0, follow from each X to $Next(X)$ until reaching $N - 1$.

Runtime Example

Nodes	0	1	2	3	4	5
Price	0	1	2	3	4	5
0		20	90	80	90	90
1			40	30	80	80
2				10	12	70
3					15	20
4						10
5						

Next(5) = 5; Costs(5) = 00;

Next(4) = 5; Costs(4) = 10;

Next(3) = 5; Costs(3) = 20; Alternative: next 4, costs 25

Next(2) = 4; Costs(2) = 22; Alternative: next 3, costs 30

Next(1) = 3; Costs(1) = 50; Alternative: next 2, costs 62

Next(0) = 1; Costs(0) = 70; Alternative: next 5, costs 90

Final Voyage: 0 -> 1 -> 3 -> 5

Java Script Implementation

```
function dynamicprogramming()  
{ var i; var j; var k;  
  next[n-1]=n-1; costs[n-1]=0;  
  for (i=n-2;i>=0;i=i-1)  
    { k = n-1;  
      for (j=n-2;j>i;j=j-1)  
        { if (price[i][j]+costs[j]<price[i][k]+costs[k])  
          { k = j; } }  
      next[i] = k; costs[i] = price[i][k]+costs[k]; } }
```

<http://www.comp.nus.edu.sg/~gem1501/dynamicprogramming.html>

Summary of Today's Lecture

- Searches and traversals
- Divide and conquer
- Greedy algorithms
- Dynamic programming

Next week

- Correctness of algorithms
 - Kinds of errors
 - Sources of errors
 - Correctness
- Efficiency of algorithms
 - Example
 - Big O
 - Recurrences