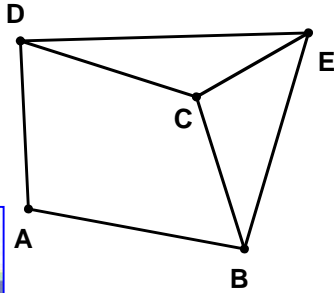
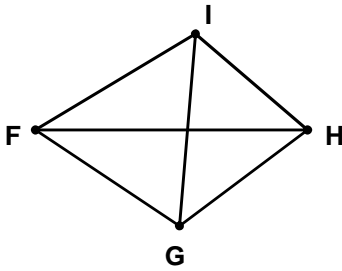



Graphs

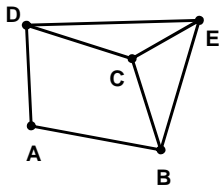
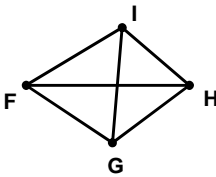
$G = (V, E)$ a graph
 $V = \{v_i\}$ set of vertices
 $E = \{e_k\}$ set of edges, $e_k = (v_i, v_j)$.



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs
10/7/2010
1

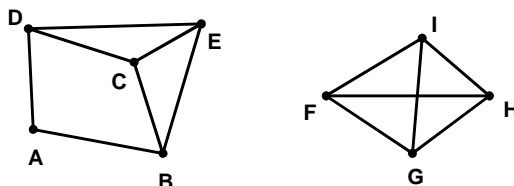
Graph definitions

<p>directed graph</p> <p>labeled graph</p> <p>edge weights</p> <p>vertex weights</p> <p>path in a graph</p> <p>connected graph</p> <p>connected component</p>	<p>edges e_k form ordered pair of vertices (v_i, v_j)</p> <p>labels given to vertices and/or edges</p> <p>number given to edge, represents for instance distance</p> <p>number given to vertex, representing for instance quality</p> <p>list of vertices, with an edge between successive vertices</p> <p>graph with a path between every pair of vertices</p> <p>maximal connected subgraph</p>
--	---

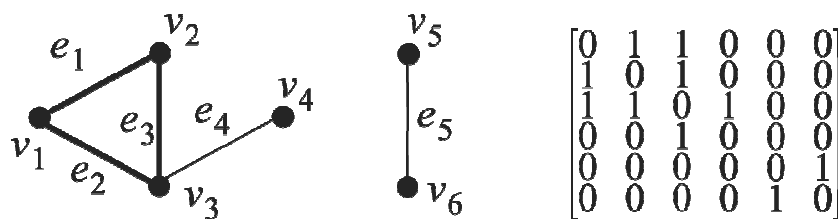
ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs
10/7/2010
2

Graph definitions (2)



- simple path** path in which vertices only occur once
- cycle** simple path, but first and last vertex identical
- tree** graph without cycles
- spanning tree** subgraph without cycles, with all vertices but possibly subset of edges
- sparse graph** graph with relatively few edges
- dense graph** graph with relatively many edges

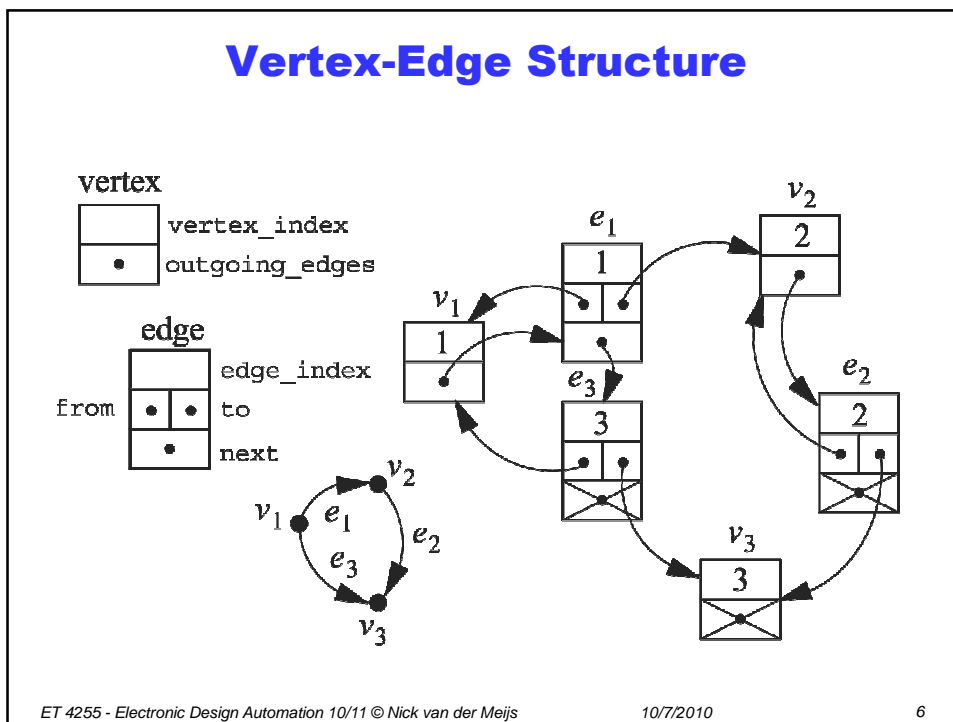
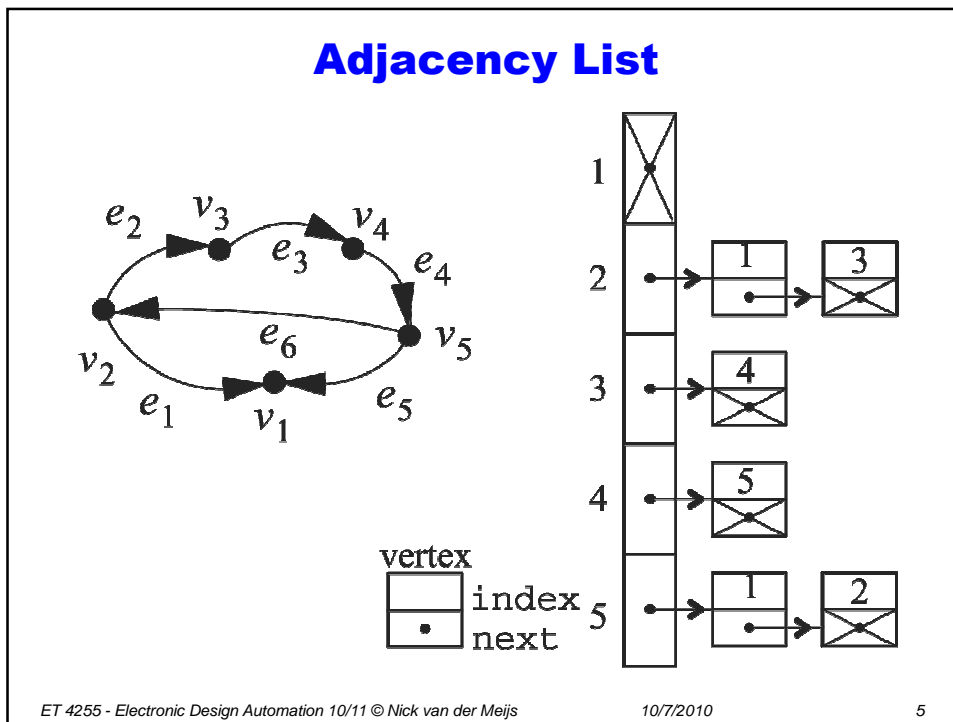
Data Structures for Graphs Adjacency Matrix



In general:

- Multiple possibilities for representing a graph
- Different characteristics of each
- Choice would depend on needs of algorithm and properties of graph (density)





Complexity of Algorithms

- Computational Complexity
- calculation time and memory occupation

Worst-case complexity

- maximum computation time (memory usage) for all inputs
- sometimes too pessimistic
- sometimes necessary

Average-case complexity

- computation times averaged over all possible inputs, taking the distribution into account
- usually much more difficult to determine (mathematics, distribution function of inputs)
- not necessarily same as worst-case: for instance quicksort



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

7

Asymptotic Complexity Analysis

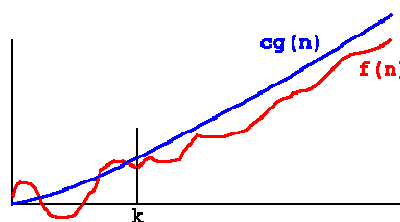
- Eliminate details of computer and implementation
- Resulting in asymptotic function of "problem size"
- Constant factors ignored

Upper Limit

$$f = f(n) \quad n \geq 0$$

$$g = g(n) \quad n \geq 0$$

$$f = O(g) \text{ if } \exists c \text{ such that } f(n) \leq c g(n) \forall n \geq k \geq 0$$



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

8

Asymptotic Complexity Analysis

- Eliminate details of computer and implementation
- Resulting in asymptotic function of "problem size"
- Constant factors ignored

Upper Limit

$$f = f(n) \quad n \geq 0$$

$$g = g(n) \quad n \geq 0$$

$$f = O(g) \text{ if } \exists c \text{ such that } f(n) \leq c g(n) \quad \forall n \geq k \geq 0$$

Lower Limit

$$f = \Omega(g) \text{ if } g = O(f)$$

Identical Asymptote

$$f = \Theta(g) \text{ if } f = O(g) \text{ and } f = \Omega(g)$$

$$f = a + bn + cn^2 \quad \Rightarrow f = O(n^2) ?$$

$$f = a + bn + cn \log n \quad \Rightarrow f = O(n \log n) ?$$

Computational Complexity Examples linear algebra

inner product

$$r = \bar{u} \cdot \bar{v}$$

$r = 0$

for $i := 1$ to N do

$$r = r + u[i] \cdot v[j]$$

complexity: $O(N)$

matrix-vector-multiplication

$$\bar{u} = P \cdot \bar{v}$$

for $i = 1$ to N do

$$u[i] = 0$$

for $j = 1$ to N do

$$u[i] = u[i] + P[i][j] \cdot v[j]$$

complexity: $O(N^2)$

matrix-matrix-multiplication complexity: $O(N^3)$

Efficiency of Algorithms

- important with growing input size
- memory efficiency often imposes hard limit
virtual memory != infinite memory

Maximum Contiguous Sum

31	-41	59	26	-53	58	97	-93	-23	84
----	-----	----	----	-----	----	----	-----	-----	----

$$\Sigma = 187$$

Given: $A_1 \dots A_N \in \mathbb{Z}$, at least one $A_i > 0$

Find i, j such that $\sum_{k=i}^j A_k$ is maximal

MCS: Algorithm 1

31	-41	59	26	-53	58	97	-93	-23	84
	↑	↑		↑					
	L	I		U					

```

MaxSoFar := 0
for L := 1 to N do
  for U := L to N do
    sum := 0
    for I := L to U do
      sum := sum + x[I]
    ← sum of x[ L ... U ]   example: 9
    MaxSoFar := max (MaxSoFar, sum)
  
```

MCS: Algorithm 2

31	-41	59	26	-53	58	97	-93	-23	84
	↑			↑					
	L			U					

```

MaxSoFar := 0
for L := 1 to N do
  sum := 0
  for U := L to N do
    sum := sum + x[U]
  ← sum of x[ L ... U ]
  MaxSoFar := max (MaxSoFar, sum)
  
```

Faster: one nested for-loop less

Analysis of Algorithm 2

```

MaxSoFar := 0
for L := 1 to N do
  sum := 0
  for U := L to N do
    sum := sum + x[U]
    ← sum of x[ L ... U ]
  MaxSoFar := max (MaxSoFar, sum)

```

- Constant time per inner loop iteration
- Count number of times inner loop is executed

- $\sum_{L=1}^N \sum_{U=L}^N O(1) = O(??)$

Analysis of Algorithm 2 (Ctd)

$$\begin{aligned}
 & \sum_{L=1}^N \sum_{U=L}^N 1 \\
 &= \sum_{L=1}^N N - L + 1 = N(N+1) - \sum_{L=1}^N L \\
 &= N(N+1) - \frac{N(N+1)}{2} \\
 &= \frac{1}{2}N + \frac{1}{2}N^2 = O(N^2)
 \end{aligned}$$

Analysis of Algorithm 1

```

MaxSoFar := 0
for L := 1 to N do
  for U := L to N do
    sum := 0
    for I := L to U do
      sum := sum + x[I]
      ← sum of x[ L ... U ]
    MaxSoFar := max (MaxSoFar, sum)
  
```

$$\sum_{L=1}^N \sum_{U=L}^N \sum_{I=L}^U 1 = \frac{1}{6}N^3 + \frac{1}{2}N^2 + \frac{1}{3}N = O(N^3)$$

Challenge: Algorithm 3

- Can you develop a linear time algorithm?
- Solution next lecture

Comparison

algorithm	1	2	3
computation time (μ sec)	$3.4 N^3$	$13 N^2$	$33 N$
computation time as a function of N	10^2	10^3	10^4
	3.4 sec	130 ms	3.3 ms
	10^3	1 hour	13 sec
	10^4	39 days	22 min
	10^5	108 years	1.5 days
	10^6	1080 centuries	5 months
			33 sec

Comparison (2)

algorithm 1, **Cray**, Fortran: $3 N^3$ nanosec
 algorithm 3, **TRS 80**, Basic: $19.5 N$ millisecc

N	alg. 1 CRAY	alg. 3 TRS 80
10	3 μ s	200 ms
100	3 ms	2 sec
10^3	3 sec	20 sec
10^4	49 min	3.2 min
10^5	35 days	32 min
10^6	95 years	5.4 hours

source: Bentley, CACM V.27 1984

Typical Complexities

$f = O(1)$	constant computation time (memory usage) for instance hashing
$\lg N$	logarithmic searching in (binary) tree of N items
N	linear maximum of N unsorted items
$N \lg N$	often with recursive divide & conquer algorithms
N^2	quadratic applying all pairs of N items
N^3	cubic matrix – matrix multiplication
2^N	exponential exhaustive search

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

21

Computation time vs Problem size

computation time (seconds), 1 μ sec per instruction

complexity	problem size (N)					
	20	50	100	200	500	1000
$1000 N$.02	.05	.1	.2	.5	1
$1000 N \lg N$.09	.3	.6	1.5	4.5	10
$100 N^2$.04	.25	1	4	25	2 min
$10 N^3$.02	1	10	1 min	21 min	2.7 hr
2^N	1	35 yr	3×10^4 century			

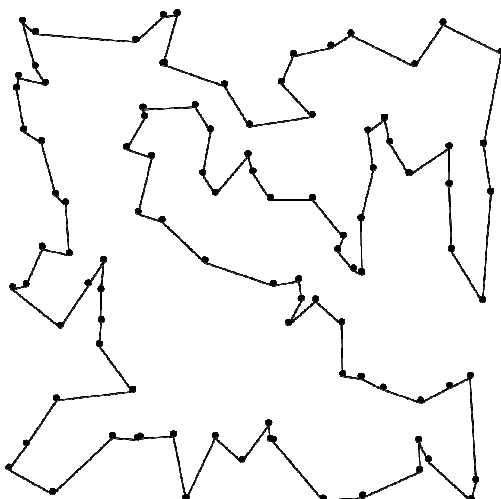
ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

22

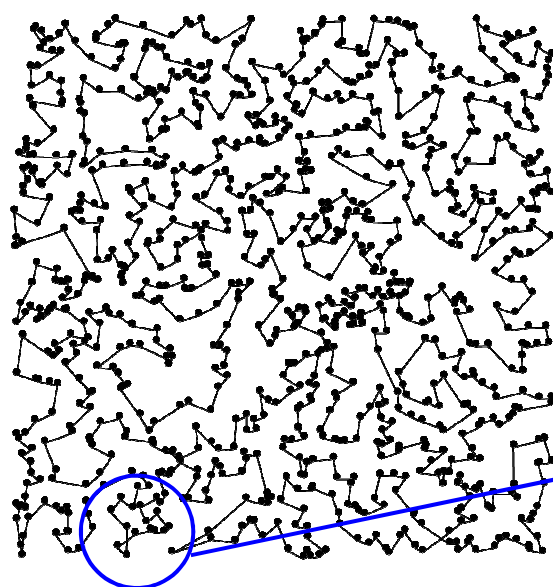
Traveling Salesman Problem

- No polynomial algorithm known
- Exact solution only by exhaustive search
- $\frac{1}{2}((N-1)!)$ possible routes

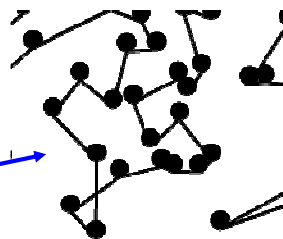


ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010

23



- Result for 1000 cities
- Clearly sub-optimal



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

24

Problem Classification

Suppose

- Π
an optimization (minimization, maximization) problem
- Π_t
derived threshold problem

Π_t is in

- class P
in case a deterministic algorithm with polynomial computation time exists.
efficient to solve
- class NP
in case a non-deterministic algorithm with polynomial computation time exists.
inefficient on deterministic machine
efficient on det. machine: checking the solution
for example: partitioning, traveling salesman

NP-complete and NP-hard problems

NP-complete

subclass of NP such that from the existence of polynomial algorithm follows that $P = NP$.

NP-hard

Π is NP-hard if derived Π_t NP-complete.
At least as hard as NP-complete

