EE 5301 – VLSI Design Automation I

Part II: Algorithms

Kia Bazargan

University of Minnesota

References and Copyright

- Textbooks referred (none required)
  - [Mic94] G. De Micheli
    "Synthesis and Optimization of Digital Circuits"
    McGraw-Hill, 1994.
  - [CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest
    "Introduction to Algorithms"
    MIT Press, 1990.
  - [Sar96] M. Sarrafzadeh, C. K. Wong
    "An Introduction to VLSI Physical Design"
    McGraw-Hill, 1996.
  - [She99] N. Sherwani
    "Algorithms For VLSI Physical Design Automation"
    Kluwer Academic Publishers, 3rd edition, 1999.

References and Copyright (cont.)

- Slides used: (*Modified by Kia when necessary*)
  - [©Sarrafzadeh] © Majid Sarrafzadeh, 2001;
    Department of Computer Science, UCLA
  - [©Sherwani] © Naveed A. Sherwani, 1992
    (companion slides to [She99])
  - [©Keutzer] © Kurt Keutzer, Dept. of EECS,
    UC-Berekeley
    http://www-cad.eecs.berkeley.edu/~niraj/ee244/index.htm
  - [©Gupta] © Rajesh Gupta
    UC-Irvine
    http://www.ics.uci.edu/~rgupta/ics280.html

Combinatorial Optimization

- Problems with discrete variables
  - Examples:
    - SORTING: given N integer numbers, write them in increasing order
    - KNAPSACK: given a bag of size S, and items $\{(s_1, w_1), (s_2, w_2), \ldots, (s_n, w_n)\}$, where s_i and w_i are the size and value of item i respectively, find a subset of items with maximum overall value that fit in the bag.
    - More examples: http://www.research.compaq.com/SRC/JCAT/
  - A problem vs. problem instance
- Problem complexity:
  - Measures of complexity
  - Complexity cases: average case, worst case
  - Tractability - solvable in polynomial time

[©Gupta]

Algorithm

- An algorithm defines a procedure for solving a computational problem
  - Examples:
    - Quick sort, bubble sort, insertion sort, heap sort
    - Dynamic programming method for the knapsack problem
- Definition of complexity
  - Run time on deterministic, sequential machines
  - Based on resources needed to implement the algorithm
    - Needs a cost model: memory, hardware/gates, communication bandwidth, etc.
    - Example: RAM model with single processor

      → running time $\propto$ # operations

[©Gupta]

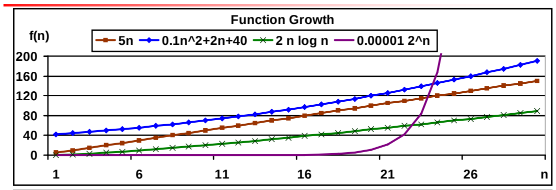
Algorithm (cont.)

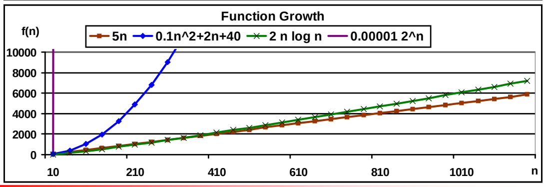
- Definition of complexity (cont.)
  - Example: Bubble Sort →
  - Scalability with respect to input size is important
    - How does the running time of an algorithm change when the input size doubles?
    - Function of input size (n). Examples: n^2+3n, 2^n, $n \log n$, ...
    - Generally, large input sizes are of interest (n > 1,000 or even n > 1,000,000)
    - What if I use a better compiler? What if I run the algorithm on a machine that is 10x faster?

```
for (j=1 ; j< N; j++) {
  for (i=; i < N-j-1; i++) {
    if (a[i] > a[i+1]) {
      hold = a[i];
      a[i] = a[i+1];
      a[i+1] = hold;
    }
  }
}
```

[©Gupta]

Function Growth Examples





Asymptotic Notions

- Idea:
  - A notion that ignores the "constants" and describes the "trend" of a function for large values of the input
- Definition
  - Big-Oh notation $f(n) = O(g(n))$
    if constants K and n_0 can be found such that:
    $\forall n \geq n_0, f(n) \leq K. g(n)$

    g is called an "upper bound" for f
    (f is "of order" g: f will not grow larger than g by more than a constant factor)

    Examples: $1/3\ n^2 = O(n^2)$ (also $O(n^3)$)
    $0.02\ n^2 + 127\ n + 1923 = O(n^2)$

Asymptotic Notions (cont.)

- Definition (cont.)
  - Big-Omega notation $f(n) = \Omega(g(n))$
    if constants K and n_0 can be found such that:
    $\forall n \geq n_0, f(n) \geq K. g(n)$

    g is called a "lower bound" for f
  - Big-Theta notation $f(n) = \Theta(g(n))$
    if g is both an upper and lower bound for f
    Describes the growth of a function more accurately than O or Ω
    Example:
    $n^3 + 4\ n \neq \Theta(n^2)$
    $4\ n^2 + 1024 = \Theta(n^2)$

Asymptotic Notions (cont.)

- How to find the order of a function?
  - Not always easy, esp if you start from an algorithm
  - Focus on the "dominant" term
    - o $4 n^3 + 100 n^2 + \log n \rightarrow O(n^3)$
    - o $n + n \log(n) \rightarrow n \log(n)$
  - $n! = K^n > n^K > \log n > \log \log n > K$
    - $\Rightarrow n > \log n, \quad n \log n > n, \quad n! > n^{10}$.
- What do asymptotic notations mean in practice?
  - If algorithm A has "time complexity" $O(n^2)$ and algorithm B has time complexity $O(n \log n)$, then algorithm B is better
  - If problem P has a lower bound of $\Omega(n \log n)$, then there is NO WAY you can find an algorithm that solves the problem in $O(n)$ time.

Problem Tractability

- Problems are classified into "easier" and "harder" categories
  - Class P: a polynomial time algorithm is known for the problem (hence, it is a tractable problem)
  - Class NP (non-deterministic polynomial time):
    ~ polynomial solution not found yet
    (probably does not exist)
    → exact (optimal) solution can be found using an algorithm with exponential time complexity
- Unfortunately, most CAD problems are NP
  - Be happy with a "reasonably good" solution
- Reading material on time complexity and NP-completeness:
  - Textbook section 3.3, Chapter 4
  - See the "Useful Links" slides at the end

Also in case anybody cares, it is incorrect to describe an optimization problem as NP-complete. Only decision problems with "Yes/No" (e.g. "does a solution exist of size K") answers can properly be termed NP-complete. Optimization problems (e.g. "find the best solution") are usually "NP-Hard". In polite company (and most journals) incorrect but well intentioned uses of "NP-complete" are accepted. -Craig Chase

[©Gupta]

Algorithm Types

- Based on quality of solution and computational effort
  - Deterministic
  - Probabilistic or randomized
  - Approximation
  - Heuristics: local search
- Problem vs. algorithm complexity

[©Gupta]

Deterministic Algorithm Types

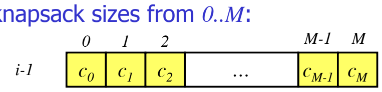
- Algorithms usually used for P problems
  - Exhaustive search! (aka exponential)
  - Dynamic programming
  - Divide & Conquer (aka hierarchical)
  - Greedy
  - Mathematical programming
  - Branch and bound
- Algorithms usually used for NP problems (not seeking "optimal solution", but a "good" one)
  - Greedy (aka heuristic)
  - Genetic algorithms
  - Simulated annealing
  - Restrict the problem to a special case that is in P

Dynamic Programming

- (read the first two examples in the document written by Michael A. Trick – see the "Useful Links (cont.)" slide)
  - Plant proposals
  - Shortest path
- 0-1 Knapsack problem:
  - Given N discrete items of size s_i and value v_i, how to fill a knapsack of size M to get the maximum value? There is only one of each item that can be either taken in whole or left out.
- Solution to the knapsack problem:
  - http://www.cee.hw.ac.uk/~alison/ds98/node122.html

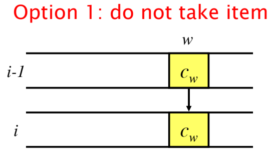
Dynamic Programming: Knapsack

- Partial solution constructed for items $1..(i-1)$ for knapsack sizes from $0..M$:

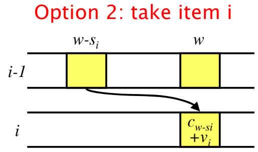


- For each knapsack size, how to extend the solution from $1..(i-1)$ to include i?

Option 1: do not take item i

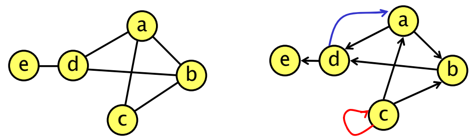


Option 2: take item i

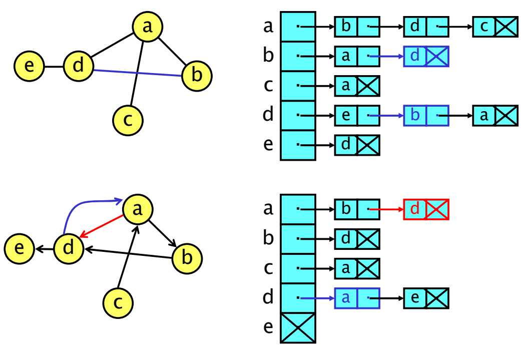


Graph Definition

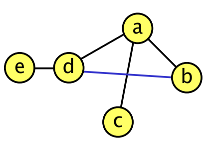
- Graph: set of "objects" and their "connections"
- Formal definition:
  - G = (V, E), $V=\{v_1, v_2, ..., v_n\}$, $E=\{e_1, e_2, ..., e_m\}$
  - V: set of vertices (nodes), E: set of edges (links, arcs)
  - Directed graph: $e_k = (v_i, v_j)$
  - Undirected graph: $e_k=\{v_i, v_j\}$
  - Weighted graph: $w: E \rightarrow R$, $w(e_k)$ is the "weight" of e_k.



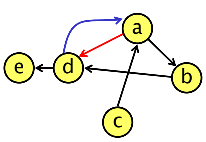
Graph Representation: Adjacency List



Graph Representation: Adjacency Matrix



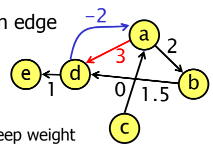
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0

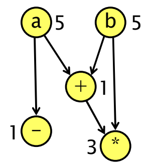


	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

Edge / Vertex Weights in Graphs

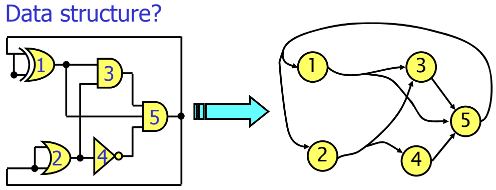
- Edge weights
  - Usually represent the "cost" of an edge
  - Examples:
    - Distance between two cities
    - Width of a data bus
  - Representation
    - Adjacency matrix: instead of 0/1, keep weight
    - Adjacency list: keep the weight in the linked list item
- Node weight
  - Usually used to enforce some "capacity" constraint
  - Examples:
    - The size of gates in a circuit
    - The delay of operations in a "data dependency graph"





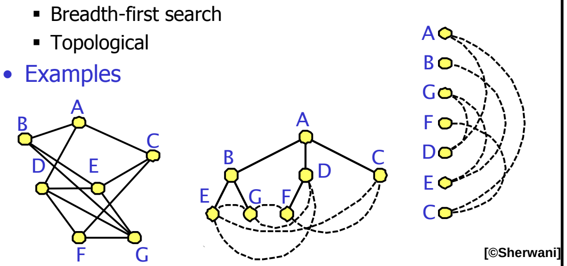
Hypergraphs

- Hypergraph definition:
  - Similar to graphs, but edges not between pairs of vertices, but between a set of vertices
  - Directed / undirected versions possible
  - Just like graphs, a node can be the source (or be connected to) multiple hyperedges
- Data structure?



Graph Search Algorithms

- Purpose: to visit all the nodes
- Algorithms
  - Depth-first search
  - Breadth-first search
  - Topological
- Examples



Depth-First Search Algorithm

```
struct vertex {
    ...
    int mark;
};

dfs ( v )
    v.marked ← 1
    print v
    for each (v, u) ∈ E
        if (u.mark != 1)   // not visited yet?
            dfs (u)


Algorithm DEPTH_FIRST_SEARCH ( V, E )
    for each v ∈ V
        v.marked ← 0   // not visited yet
    for each v ∈ V
        if (v.marked == 0)
            dfs (v)
```


Breadth-First Search Algorithm

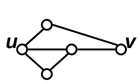
```
bfs ( v , Q)
    v.marked ← 1
    for each (v, u) ∈ E
        if (u.mark != 1)   // not visited yet?
            Q ← Q + u


Algorithm BREADTH_FIRST_SEARCH ( V, E )
    Q ← {v0}   // an arbitrary node
    while Q != {}
        v ← Q.pop()
        if (v.marked != 1)
            print v
            bfs (v)   // explore successors
```

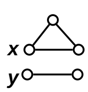
There is something wrong with this code. What is it?

Distance in (non-weighted) Graphs

- **Distance** $d_G(u,v)$:
  - Length of a shortest *u*--*v* path in *G*.



$d(u,v)=2$



$d(x,y) = \infty$

Moor's Breadth-First Search Algorithm

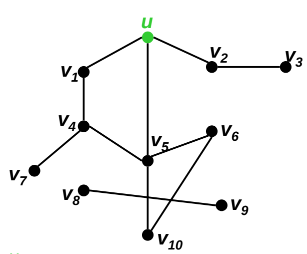
- Objective:
  - Find *d(u,v)* for a given pair (u,v) and a shortest path u--v
- How does it work?
  - Do BFS, and assign λ(w) the first time you visit a node. λ(w)=depth in BFS.
- Data structure
  - Q a queue containing vertices to be visited
  - λ(w) length of the shortest path u--w (initially ∞)
  - π(w) parent node of w on u--w

Moor's Breadth-First Search Algorithm

- Algorithm:

1. Initialize
   - o λ(w) ← ∞ for w≠u
   - o λ(u) ← 0
   - o Q ← Q+u
2. If Q ≠ ϕ, x ← pop(Q)
   else stop: "no path u--v"
3. ∀(x,y)∈*E*,
   - o if λ(y)=∞, π(y) ← x
   - o λ(y) ← λ(x)+1
   - o Q ← Q+y
4. If λ(v)=∞ return to step 2
5. Follow π(w) pointers from v to u.

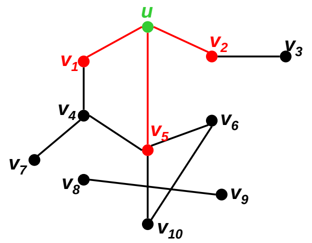
Moor's Breadth-First Search Algorithm



Q = u

Node	u	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
λ	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
π	-	-	-	-	-	-	-	-	-	-	-

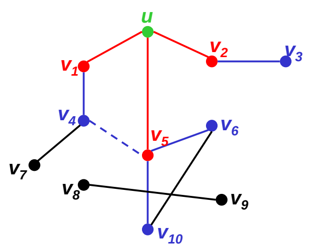
Moor's Breadth-First Search Algorithm



Q = v_1, v_2, v_5

Node	u	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
λ	0	1	1	∞	∞	1	∞	∞	∞	∞	∞
π	-	u	u	-	-	u	-	-	-	-	-

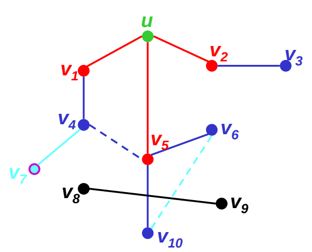
Moor's Breadth-First Search Algorithm



Q = v_4, v_3, v_6, v_{10}

Node	u	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
λ	0	1	1	2	2	1	2	∞	∞	∞	2
π	-	u	u	v_2	v_1	u	v_5	-	-	-	v_5

Moor's Breadth-First Search Algorithm



Q = v_7

Node	u	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
λ	0	1	1	2	2	1	2	3	∞	∞	2
π	-	u	u	v_2	v_1	u	v_5	v_4	-	-	v_5

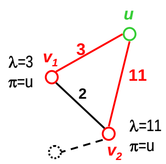
VLSI Design Automation I – © Kia Bazargan

Notes on Moor's Algorithm

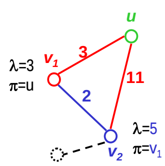
- Why the problem of BREADTH_FIRST_SEARCH algorithm does not exist here?
- Time complexity?
- Space complexity?

Distance in Weighted Graphs

- Why a simple BFS doesn't work any more?
  - Your locally best solution is not your globally best one
  - First, v_1 and v_2 will be visited



v_2 should be revisited



Dijkstra's Algorithm

- Objective:
  - Find *d(u,v)* for all pairs (u,v) (fixed u) and the corresponding shortest paths u--v
- How does it work?
  - Start from the source, augment the set of nodes whose shortest path is found.
  - decrease λ(w) from ∞ to *d(u,v)* as you find shorter distances to w. π(w) changed accordingly.
- Data structure:
  - S the set of nodes whose *d(u,v)* is found
  - λ(w) current length of the shortest path u--w
  - π(w) current parent node of w on u—w

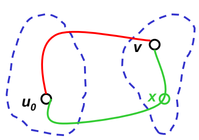
VLSI Design Automation I – © Kia Bazargan

Dijkstra's Algorithm

- Algorithm:
  1. Initialize
     - $\lambda(v) \leftarrow \infty$ for $v \neq u$
     - $\lambda(u) \leftarrow 0$
     - $S \leftarrow \{u\}$
  2. For each $v \in S'$ s.t. $u_i v \in E$,
     - If $\lambda(v) > \lambda(u_i) + w(u_i v)$,
       - $\lambda(v) \leftarrow \lambda(u_i) + w(u_i v)$
       - $\pi(v) \leftarrow u_i$
  3. Find $m=\min\{\lambda(v) | v \in S'\}$ and $\lambda(v_j)==m$
     - $S \leftarrow S \cup \{v_j\}$
  4. If $|S|<|V|$, goto step 2

Dijkstra's Algorithm - why does it work?

- Proof by contradiction
  - Suppose v is the first node being added to S such that $\lambda(v) > d(u_0, v)$ ($d(u_0, v)$ is the "real" shortest u_0--v path)
  - The assumption that $\lambda(v)$ and $d(u_0, v)$ are different, means there are different paths with lengths $\lambda(v)$ and $d(u_0, v)$
  - Consider the path that has length $d(u_0, v)$. Let x be the first node in S' on this path
  - $d(u_0, v) < \lambda(v)$, $d(u_0, v) \geq \lambda(x) + \alpha$ => $\lambda(x) < \lambda(v)$ => contradiction

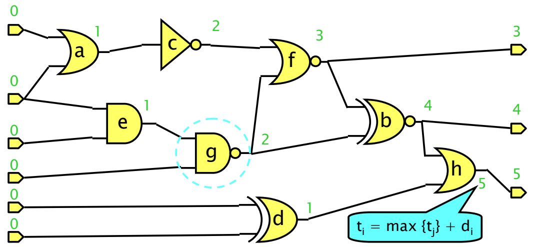


Static Timing Analysis

- Finding the longest path in a general graph is NP-hard, even when edges are not weighted
- Polynomial for DAG (directed acyclic graphs)
- In circuit graphs, "static timing analysis (STA)"…
  - …refers to the problem of finding the max delay from the input pins of the circuit (esp nodes) to each gate
  - Max delay of the output pins determines clock period
  - In sequential circuits, FF input acts as output pin, FF output acts as input pin
  - Critical path is a path with max delay among all paths
  - In addition to the "arrival time" of each node, we are interested in knowing the "slack" of each node / edge

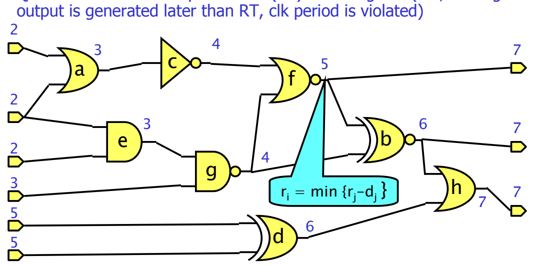
STA Example: Arrival Times

- Assumptions:
  - All inputs arrive at time 0
  - All gate delays = 1
  - All wire delays = 0
- Question: Arrival time of each gate? Circuit delay?



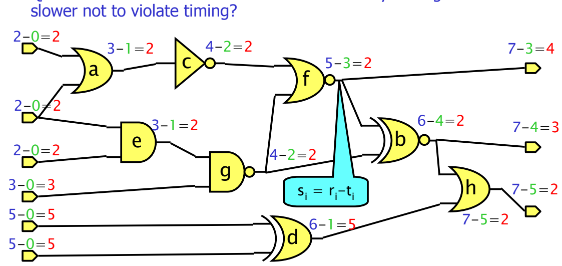
STA Example: Required Times

- Assumptions:
  - All inputs arrive at time 0
  - All gate delays = 1, wire delay = 0
  - Clock period = 7
- Question: maximum required time (RT) of each gate? (i.e., if the gate output is generated later than RT, clk period is violated)



STA Example: Slack

- Assumptions:
  - All inputs arrive at time 0
  - All gate delays = 1, wire delay = 0
  - Clock period = 7
- Question: What is the maximum amount of delay each gate can be slower not to violate timing?



STA: Issues

- STA can be done in linear time
  - How to implement?
- What would change if wires have non-zero delays?
- If the delay of one gate changes, what is the time complexity of updating the slack of all nodes?
- How can slack be used?
- How to distribute a path's slack to different edges? (the budgeting problem)
- How to maintain a list of K-most critical paths?
  - Why important?
  - Variation: paths of delay > D
  - What is the upper bound on the number of such paths?

Minimum Spanning Tree (MST)

- Tree (usually undirected):
  - Connected graph with no cycles
  - |E| = |V| - 1
- Spanning tree
  - Connected subgraph that covers all vertices
  - If the original graph not tree, graph has several spanning trees
- Minimum spanning tree
  - Spanning tree with minimum sum of edge weights (among all spanning trees)
  - Example: build a railway system to connect N cities, with the smallest total length of the railroad

Difference Between MST and Shortest Path

- Why can't Dijkstra solve the MST problem?
  - Shortest path: min sum of edge weight to individual nodes
  - MST: min sum of TOTAL edge weights that connect all vertices
- Proposal:
  - Pick any vertex, run Dijkstra and note the paths to all nodes (prove no cycles created)
- Debunk: show a counter example

Minimum Spanning Tree Algorithms

- Basic idea:
  - Start from a vertex (or edge), and expand the tree, avoiding loops (i.e., add a "safe" edge)
  - Pick the minimum weight edge at each step
- Known algorithms
  - Prim: start from a vertex, expand the connected tree
  - Kruskal: start with the min weight edge, add min weight edges while avoiding cycles (build a forest of small trees, merge them)

Prim's Algorithm for MST

- Data structure:
  - S set of nodes added to the tree so far
  - S' set of nodes not added to the tree yet
  - T the edges of the MST built so far
  - $\lambda(w)$ current length of the shortest edge (v, w) that connects w to the current tree
  - $\pi(w)$ potential parent node of w in the final MST (current parent that connects w to the current tree)

Prim's Algorithm

- Initialize S, S' and T
  - $S \leftarrow \{u_0\}$, $S' \leftarrow V \setminus \{u_0\}$ // u_0 is any vertex
  - $T \leftarrow \{\ \}$
  - $\forall v \in S'$, $\lambda(v) \leftarrow \infty$
- Initialize λ and π for the vertices adjacent to u_0
  - For each $v \in S'$ s.t. $(u_0, v) \in E$,
    - $\lambda(v) \leftarrow \omega((u_0, v))$
    - $\pi(v) \leftarrow u_0$
- While ($S' \; != \phi$)
  - Find $u \in S'$, s.t. $\forall v \in S'$, $\lambda(u) \leq \lambda(v)$
  - $S \leftarrow S \cup \{u\}$, $S' \leftarrow S' \setminus \{u\}$, $T \leftarrow T \cup \{(\pi(u), u)\}$
  - For each v s.t. $(u, v) \in E$,
    - If $\lambda(v) > \omega((u,v))$ then
      $\lambda(v) \leftarrow \omega((u,v))$
      $\pi(v) \leftarrow u$

Other Graph Algorithms of Interest...

- Min-cut partitioning
- Graph coloring
- Maximum clique, independent set
- Min-cut algorithms
- Steiner tree
- Matching
- ...

Useful Links (Also Linked from Course WebPage)

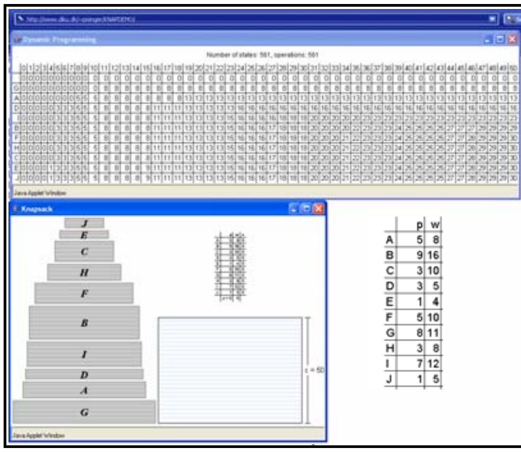
- Algorithms and Visualization
  - Compaq's JCAT: allows users to run a number of algorithms in their web browsers and visualize the progress of the program. http://www.research.compaq.com/SRC/JCAT/
  - SGI's Standard Template Library (click on the "Index" link. "Table of contents" is very useful too). http://www.sgi.com/tech/stl/
  - Microsoft MSDN library (If you don't know where to go, type "fopen" in the "Search for" textbox and click "GO" for normal C functions, and then navigate using the tree on the left. For STL documentation, search for "vector::push_back".) http://msdn.microsoft.com/library/default.asp
- Books on STL and templtes (thanks to Arvind Karandikar @ U of M for suggesting thebooks) :
  - Nicolai M. Josuttis, "The C++ Standard Library: A Tutorial and Reference", Addison-Wesley, 1999, ISBN: 0-201-37926-0.
  - David Vanevoorde and Nicolai M. Josuttis, "C++ Templates, the Complete Guide", Addison-Wesley, 2003, ISBN: 0-201-73484-2.

Useful Links (cont.)

- Time Complexity and Asymptotic Notations
  - www.cs.sunysb.edu/~skiena/548/lectures/lecture2.ps
  - Asymptotic bounds: definitions and theorems http://userpages.umbc.edu/~anastasi/Courses/341/Spr00/Lectures/Asymptotic/asymptotic/asymptotic.html
  - www.cs.yorku.ca/~ruppert/6115W-01/bigO.ps
  - CMSC 341 (at CSEE/UMBC) Lecture 2 http://www.csee.umbc.edu/courses/undergraduate/CMSC341/fall02/Lectures/AA/AsymptoticAnalysis.ppt
  - Michael A. Trick, "A Tutorial on Dynamic Programming", http://mat.gsia.cmu.edu/classes/dynamic/dynamic.html

Papers

- C. J. Alpert, A. Devgan, S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation", Design Automation Conference, pp. 479–484, 1999.
  - Shows that the Elmore model overesimates delay, offers a new, more accurate model. Uses this model to optimize the buffer insertion



	p	w
A	5	8
B	9	16
C	3	10
D	3	5
E	1	4
F	5	10
G	8	11
H	3	8
I	7	12
J	1	5

From: http://www.diku.dk/~pisinger/KNAPDEMO/
[Dynamic programming, data.3, primal, table, no bounds]