ID1020 2018-10-11 #14

How to model both vertex and edge weights?
- split node into two (in/out)
- add weight to edge between them.

Multiple sources/sinks?
- Add node connected to them

Real-time systems
- Job
  - Start time
  - critical path (longest exec. time)
  - deadline

Longest paths in edge-weighted DAGs
- Formulate as a shortest path
  - Negate all weights
  - Calculate shortest path
  - Negate result.
- Used for parallel job scheduling

Negative weights
- Dijkstra does not work
- Negative cycle: directed cycle, sum of edge weights <0.
  - weight is lowered for each cycle
- Bellman-Ford algorithm
  - Initialize distTo[s] = 0 and $distTo_0[v] = \infty$ for all other vertices
  - Repeat V times
    - Relax all edges
  - Quadratic in theory
  - can be terminated when nothing changes.

Dynamic programming algorithm computes SPT in time proportional to $V \times E$

Divide-and-Conquer: Break large problems into smaller subproblems. Combine the smaller solutions

Dynamic programming: Remember past results to find new results. Generally used for optimization where you need to find the best of multiple solutions.

Greedy algorithm: Make choices that seem to be the best right there and then and hope that it will lead you on the right path

Eager algorithm: Same as greedy

Bellman-Ford: If distTo[v] does not change during pass i, no need to relax any edges pointing from v in pass i+1

If negative cycles exist, Bellman-Ford gets stuck in loop.

Finding negative cycle:
If any vertex v is updated in pass V, there exists a negative cycle (and we can trace back edgeTo[v] entries to find it)

Arbitrage problem can be described as a negative cycle problem.

(Vertex = currency, Edge = transaction)

Remaining part (not covered in book)

CFG (Context free grammars): Defined by a set of production rules.
- see presentation

Parsing, building parse tree

Halting problem
- Can we write a program which can determine if a given program will terminate.

Not solvable:
Proof sketch: if(halts()) while(true);

Master theorem (not on exam)
- How can we derive complexity of recursive methods?
- Repeated substitution.

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be a function over the positive numbers defined by the recurrence.

$T(n) = aT(n/b) + f(n)$. If $f(n) = \Theta(n^d)$, where $d \geq 0$, then
- $T(n) = \Theta(n^d)$ if $a < b^d$
- $T(n) = \Theta(n^d \log n)$ if $a = b^d$
- $T(n) = \Theta(n^{\log_b a})$ if $a > b^d$

Sustainibility

Efficient algorithms:
- saves energy
- Allows longer life span for computers, smart phones, tablets...
- Allows less costly solutions to problems
- May give more people access to information.