Today’s Lecture

• Algorithm generalities / complexity

• Directed graphs, WDAGs
Genomes are big but computers are fast!

- Typical laptop clock speed: ~ 1 Ghz
  - Potentially billions of CPU instructions / sec
- Important practical consideration in dealing with genome-scale data sets: compared to CPU operations,
  - non-cache memory accesses are very slow (100s of cycles)
  - disk accesses are even slower (1000s of cycles)
  - for both, random (non-sequential) accesses are much slower than sequential accesses
Algorithms – Some General Remarks

• The most widely used algorithms are the oldest
  – e.g. sorting lists, traversing trees, dynamic programming.

The challenge in CMB is usually *not* finding *new* algorithms, but rather
  – finding *biologically appropriate applications* of old ones.

• Often prefer
  – suboptimal but easy-to-program algorithm over more optimal one
  – or space-efficient algorithm over time-efficient one.

• *Probabilities* are important in
  – interpreting results
  – guiding search

The most powerful analyses generally involve probabilistic models, rather than deterministic ones.
Algorithmic Complexity

• Basic questions about an algorithm:
  – how long does it take to run?
  – how much space (RAM or disk space) does it require?

• Would like precise function $f(N)$, e.g.
  
  $$f(N) = .05 \, N^3 + 50.7 \, N^2 + 6.03 \, N$$

  for
  – running time in secs, or
  – space in kbytes,

  as function of the size $N$ of input data set.

• But
  – tedious to derive &
  – depends on (often uninteresting – though important!) hardware &
    software implementation details.
Algorithmic Complexity (cont’d)

• Instead, more customary to give “the” asymptotic complexity, i.e. expression $g(N)$ such that

$$C_1 g(N) < f(N) < C_2 g(N)$$

for some constants $C_1$ and $C_2$, and $N$ large enough.

• This is written $O(g(N))$, where notation $O()$ means “up to an unspecified multiplicative constant”.
  – e.g. for the $f(N)$ above, the dominating term for large $N$ is $.05 N^3$, so
    • can take $g(N) = N^3$
    • asymptotic complexity = $O(N^3)$. 
Algorithmic Complexity (cont’d)

• Can be misleading, since
  – for small $N$ a different term may dominate
    • (e.g. $2^d$ term in above example much more important for $N < 1000$)
  – size of constant may be quite important
    • (big difference between .05 and 5,000,000!)
    • e.g. BLAST and Smith-Waterman both $O(N^2)$, but size of constant enormously different

• *but* very useful as rough guide to performance.
Algorithmic Complexity (cont’d)

- Cache misses (non-cache memory accesses) and disk accesses often dominate running time, yet are ‘invisible’ to complexity analysis (because affect constant factor only)
Algorithmic Complexity (cont’d)

• Another limitation to complexity analysis:
  – time or space requirement may depend on specific characteristics of input data.

• Usually give “worst case” complexity
  – applies to the worst data set of a given size,

*but*

  – in biological situations the *average biologically occurring case* is
    • more relevant
    • often much easier than worst case (which may never arise in practice), or even “average case” in some idealized sense.
Algorithmic Complexity (cont’d)

• Proof that a problem is *NP-hard*
  – (has complexity very likely greater than any polynomial function of $N$ and therefore effectively unsolvable for large $N$)
  
  can be useful in guiding search for more efficient algorithms

*but* can also be misleading, since

– we need *some* solution anyway, for data sets occurring in practice

– average *biologically relevant* case may be quite manageable
Directed Graphs

• A directed graph is a pair \((V, E)\) where
  – \(V\) is a finite set of vertices, or nodes.
  – \(E\) is a set of ordered pairs (called edges) of vertices in \(V\).

• An edge \((v_i, v_j)\) is said to leave \(v_i\) and to enter \(v_j\).
  – \((v_i \text{ and } v_j \text{ are vertices})\)

• in-degree of a vertex = \# edges entering it;
• out-degree = \# edges leaving it.
Example:

- \( V = \{1,2,3,4,5,6\} \),
- \( E = \{(1,2), (1,3), (2,4), (4,1), (5,3), (3,1)\} \)
- Vertex 3 has in-degree 2 and out-degree 1.
Paths and Cycles

• A *path* of length $k$ in $G$ from $u$ to $u'$ (vertices) is
  – a sequence $P$ of vertices $(v_0, v_1, \ldots, v_k)$ such that
    • $v_0 = u$,
    • $v_k = u'$, and
    • $(v_{i-1}, v_i)$ is an edge for $i = 1, 2, \ldots, k$.

• A path can have length 0.
• We write $|P| = k$.
• A *cycle* is a path of length $\geq 1$ from a vertex to itself.

• In example at right,
  – $(1,2,4)$ is a path,
  – $(1,3,5)$ is not, and
  – $(1,2,4,1)$ and $(1,3,1)$ are cycles.
Paths and Cycles (cont’d)

• Can join
  – any path \((u, \ldots, v)\) from \(u\) to \(v\), to
  – any path \((v, \ldots, w)\) from \(v\) to \(w\)

  to get a path \((u, \ldots, v, \ldots, w)\) from \(u\) to \(w\).
DAGs

- A *directed acyclic graph* (DAG) is a directed graph with no cycles.
- In a DAG, for distinct nodes $v_i$ and $v_j$, we say
  - $v_i$ is a *parent* of $v_j$, and $v_j$ is a *child* of $v_i$, if
    - there is an edge $(v_i, v_j)$
  - $v_i$ is an *ancestor* of $v_j$, and $v_j$ is a *descendant* of $v_i$, if
    - there is a path from $v_i$ to $v_j$
- In a DAG the length of a path cannot exceed $|V| - 1$, because
  - in a path of length $\geq |V|$, at least one vertex $v$ would have to appear twice in the path;
  - but then there would be a path from $v$ to $v$, i.e. a cycle.
Structure of DAGs

• Define the *depth* of a node $v$ in $V$ as:
  – the length of the longest path ending at $v$;

by above, the depth is well-defined and $\leq |V| - 1$.

• *Every descendant $w$ of a node $v$ has higher depth than $v$*: If
  – $(u, \ldots, v)$ is path of length $n = \text{depth}(v)$ ending at $v$, and
  – $(v, \ldots, w)$ is path from $v$ to $w$,

then $(u, \ldots, v, \ldots, w)$ is a path of length $> n$ ending at $w$, so $\text{depth}(w) > n$. 
Structure of DAGs (cont’d)

• **Every node v of positive depth has a parent of depth exactly one less:**
  – Let \((u, \ldots, v', v)\) be path of length \(n = \text{depth}(v)\) ending at \(v\).
  – Then \(v'\) is a parent of \(v\).
  – Since \((u, \ldots, v')\) has length \(n - 1\), \(\text{depth}(v') \geq n - 1\).
  – Since also \(\text{depth}(v') < n\) (because \(v\) is a descendant of \(v'\)), \(\text{depth}(v')\) is exactly \(n - 1\).

• **The nodes on any path are of increasing depth.**
Structure of DAGs (cont’d)
Important special cases:

- A *(rooted)* **tree** is a DAG which
  - has unique depth 0 node (the *root*), *and*
  - every other node has in-degree 1
    - *i.e.* has a unique parent, of depth one less than that of the node).

- A **binary tree** is a tree in which
  - every node has out-degree at most 2.

- A **linked list** is a tree in which
  - every node has out-degree at most 1
  - or equivalently, a DAG in which ∃ at most one node of each depth
binary tree

linked list

\[
\begin{array}{c}
\text{binary tree} \\
\text{linked list}
\end{array}
\]
Remarks on Depth Structure

• For *dynamic programming* algorithm
  – we need an order $v_1, v_2, ..., v_n$ for the vertices
    • (not a path!)
      in which parents appear before children.
  – From the above, *depth order*
    • (in which depth 0 nodes are listed first, then depth 1 nodes, etc.)
      is such an order.
  – In general there are many other such orders.

• We haven’t given constructive procedure for finding the depths of all vertices.
  – For an arbitrary DAG, can be done in $O(|V| + |E|)$ time;
  – we omit algorithm, since for DAGs related to sequence analysis, the depth structure is obvious.
Weighted Directed Graphs

• A *weighted directed graph* is
  – a directed graph \((V, E)\) together with
  – a function \(w\) from \(E\) to the real numbers,
    • i.e. with a numerical *weight* \(w(e)\) (which may be positive, negative, or 0) associated to each edge \(e\).

A weighted DAG is called a WDAG.

• The *(sum)* *weight of a path* is defined to be the sum of the weights on the edges of the path.

• Similarly, the *product weight of a path* is the product of the edge weights
  – usually only consider this when all weights are non-negative.

• weight of a path \(P\) is written \(w(P)\)

• For a path of length 0 (i.e. consisting of a single vertex):
  – the sum weight is 0
  – the product weight is 1