Objectives

Oh, the places you've been!

Oh, the places you'll go!

> Now, everything comes down to expert knowledge of **algorithms** and **data structures**. If you don't speak fluent **O-notation**, you may have trouble getting your next job at the technology companies in the forefront.
> -- Larry Freeman

Algorithm Design Patterns

What are some approaches to solving problems?

How do they compare in terms of difficulty?

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming

Course Objectives: Given a problem…

You'll recognize when to try an approach
- AND, when to bail out and try something different

Know the steps to solve the problem using the approach
- e.g., breaking it into subproblems, sorting possibilities in some order

Know how to ***analyze*** the run time of the solution
- e.g., solving recurrence relation

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality – Chapter 7
- Reductions – Chapter 8
- Local search – Chapter 12
- Randomization – Chapter 13

What Was Our Goal In Finding a Solution?

Polynomial Time → Efficient

POLYNOMIAL-TIME REDUCTIONS

Classify Problems According to Computational Requirements

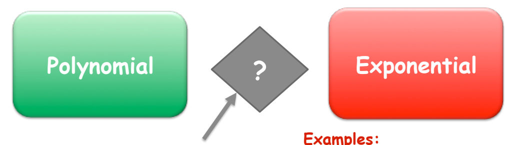
Q. Which problems will we be able to solve in practice?

A. Working definition. [Cobham 1964, Edmonds 1965, Rabin 1966] Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Classify Problems

Classify problems according to those that can be solved in polynomial-time and those that cannot.



Frustrating news: *Many* problems have defied classification.

Chapter 8. Show that problems are "computationally equivalent" and appear to be manifestations of one *really hard* problem.

Examples:
- *Given a Turing machine, does it halt in at most k steps?*
- *Given a board position in an n-by-n generalization of chess, can black guarantee a win?*

Polynomial-Time Reduction

Suppose we could solve Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem *X* polynomial reduces to problem *Y* if arbitrary instances of problem *X* can be solved using:

- Polynomial number of standard computational steps, *plus*
- Polynomial number of calls to **oracle** that solves problem Y
  - Assume have a black box that can solve Y

Notation. $X \le_P Y$

"X is polynomial-time reducible to Y"

Conclusion. If X can be solved in polynomial time and $Y \le_P X$, then Y can be solved in polynomial time.

NP Complete Problems

Problems from many different domains whose complexity is unknown

NP-completeness and proof that all problems are equivalent is **POWERFUL**!

- All open complexity questions are really ONE open question

What does this mean?

- "Computationally hard for practical purposes but we can't prove it"
- If you find an NP-Complete problem, you can stop looking for an efficient solution
  - Or figure out efficient solution for ALL NP-complete problems

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \le_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \le_P Y$ and Y cannot be solved in polynomial-time, then X cannot be solved in polynomial time.

Establish equivalence. If $X \le_P Y$ and $Y \le_P X$, we use notation $X \equiv_P Y$.

Basic Reduction Strategies

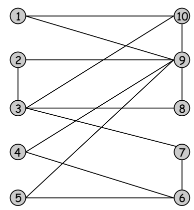
Reduction by simple equivalence

Reduction from special case to general case

Reduction by encoding with gadgets

Independent Set

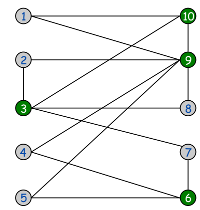
Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≥ *k*, and for each edge at most one of its endpoints is in *S*?



Ex. Is there an independent set of size ≥ 6?
Ex. Is there an independent set of size ≥ 7?

Independent Set

Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≥ *k*, and for each edge at most one of its endpoints is in *S*?

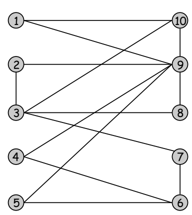


Ex. Is there an independent set of size ≥ 6? Yes
Ex. Is there an independent set of size ≥ 7? No

independent set

Vertex Cover

Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≤ *k*, and for each edge, at least one of its endpoints is in S?

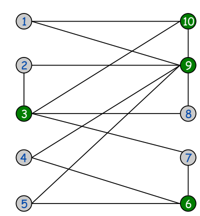


Application: place guards within an art gallery so that all corridors are visible at any time

Ex. Is there a vertex cover of size ≤ 4?
Ex. Is there a vertex cover of size ≤ 3?

Vertex Cover

Given a graph G = (V, E) and an integer *k*, is there a subset of vertices S ⊆ V such that |S| ≤ *k*, and for each edge, at least one of its endpoints is in S?



Ex. Is there a vertex cover of size ≤ 4? Yes
Ex. Is there a vertex cover of size ≤ 3? No

vertex cover

Problem

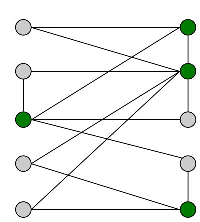
Not known if either Independent Set or Vertex Cover can be solved in polynomial time

BUT, what can we say about their relative difficulty?

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET

Pf. We show *S* is an independent set iff V - S is a vertex cover



independent set
vertex cover

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET

Pf. We show S is an independent set iff V - S is a vertex cover

$\Rightarrow$

- Let S be any independent set
- Consider an arbitrary edge (u, v)
- Since S is an independent set $\Rightarrow$ $u \notin S$ or $v \notin S$ $\Rightarrow$ $u \in V - S$ or $v \in V - S$
- Thus, V - S covers (u, v)
  - Every edge has one end in V-S

$\rightarrow$ V-S is a vertex Cover

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET

Pf. We show S is an independent set iff V - S is a vertex cover

$\Leftarrow$

- Let V - S be any vertex cover
- Consider two nodes $u \in S$ and $v \in S$
- Observe that $(u, v) \notin E$ since V - S is a vertex cover
- Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set

Reduction from Special Case to General Case

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

REVIEWING SOLUTIONS/ EXPECTATIONS

Finding Median of Two Sorted Lists

Find the median of two sorted lists

Median: half the elements are bigger and half the elements are smaller

A	1	4	5	7	8	9	11	12
B	-2	-1	0	2	3	6	10	13

If median falls at position *k* in A, it's at *n-k* in B

Finding Median of Two Sorted Lists

Compare medians of lists

- $k = \lceil n/2 \rceil$

A	1	4	5	7	8	9	11	12
B	-2	-1	0	2	3	6	10	13

If A[k] < B[k], B[k] > the first *k* elements of A

- B[k] is $2k^{th}$ element; can ignore 2^{nd} half of B

Finding Median of Two Sorted Lists

Compare medians of remaining lists

A	1	4	5	7	8	9	11	12
B	-2	-1	0	2	3	6	10	13

If A[k] < B[j], B[j] > the first *k* elements of A

- B[k] is k+jth element; can ignore 2nd half of B

Finding Median of Two Sorted Lists

Compare medians of remaining lists

A	1	4	5	7	8	9	11	12
B	-2	-1	0	2	3	6	10	13

If A[k] < B[j], B[j] > the first *k* elements of A

- B[k] is 2kth element; can ignore 2nd half of B

Finding Median of Two Sorted Lists

Compare medians of remaining lists

A	1	4	5	7	8	9	11	12
B	-2	-1	0	2	3	6	10	13

When down to 1 element in each list, median is average of 2

Analyzing Algorithm

Recurrence Relation: T(n) = T(n/2) + O(1)

- Reduce problem to one of half the size
- Comparison takes constant time

T(n) = O(log n)

Independent Sets

Goal: Maximize value of independent set

- Keep track in M

Break problem into subproblems

- For node j, either pick the node
  - Which means can't pick next node
  - Opt(j) = w_j + Opt(j-2)
- Or don't pick node j
  - Get the best solution for the remaining nodes
  - Opt(j) = Opt(j-1)

Independent Sets

Find max value:

- M[0]=0
- For each node j =1 to n
  - M[j] = max(w[j]+M[j-2], M[j-1])

Find independent set: Trace backwards through M

- FindSolution(j):
  - If j == 0: return
  - If M[j] == w[j] + M[j-2]
    - add j to independent set
    - FindSolution(j-2)
  - Else:
    - FindSolution(j-1)

Final

Available Saturday at 2 p.m.

Due Friday at 5 p.m.

Open lecture notes, your notes, book

Unlimited time (until next Friday at 5 p.m.)