Recall the plan for this course:

I. Design of Algorithms

II. Analysis of Algorithms

III. Lower Bounds — do we have the <u>best</u> algorithm?

the final section of the course

Suppose we have an algorithm *A* for problem *P* with runtime $T(n)$.

Is algorithm *A* the best?

e.g. Branch and Bound for Max Independent Set has run time $\Theta(2^n)$ — is that the best algorithm for Max Independent Set?

We would need to show that <u>any</u> algorithm for Max Independent Set has worst case runtime at least 2^n (asymptotically). Such lower bounds are hard to prove.

State of the Art in Lower Bounds/Impossibility Results

- some problems don't have algorithms. Proved by Alan Turing, 1930's. We will cover this at the end of the course. Also in CS 245, CS 360.

- some problems can only be solved in exponential time.

- there are some fine-grained lower bounds, e.g. you have seen an $n \log n$ lower bound for sorting, but that was only for a restricted model of computing (comparisons only).

Major Open Question

There are many problems, e.g. Travelling Salesman, 0-1 Knapsack, where no one knows a polynomial time algorithm and no one can prove there's no polynomial time algorithm.
The best we can do: prove that a large set of problems are ***equivalent*** in the sense that a poly. time algorithm for one yields poly. time algorithms for all

Our focus is on ***polynomial time*** and the class P.

Our main tool is ***reductions***.

We will spend some time defining these.

The class of equivalent problems are the ***NP-complete problems***.

Goals:

- be familiar with the concept of NP-completeness
- recognize some NP-complete problems
- do some NP-completeness proofs

Polynomial time

Definition. An algorithm runs in polynomial time if its runtime (asymptotic, worst-case) is $O(\ n^k\ )$ where n is the input size and k is a constant.

Examples.

polynomial time ?

$$O(n^2),\ \ O(n^5),\ \ O(n \log n),\ \ O(2^n),\ \ O(n!),\ \ O(n^{1{,}000{,}000}),\ \ O(n^{2^{2^2}})$$

$$\checkmark \qquad \checkmark \qquad \checkmark \qquad \times \qquad \times \qquad \checkmark \qquad \checkmark$$

Most of the algorithms we've studied have been polynomial time, except for backtracking, branch-and-bound, and the pseudo-polynomial time algorithm for the 0-1 knapsack problem, with runtime $O(nW\ )$.

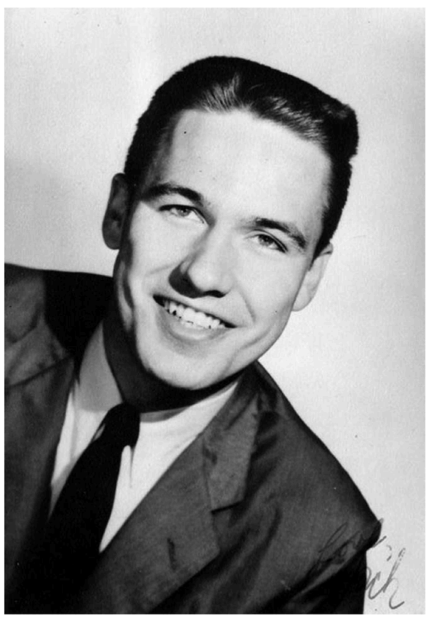
Polynomial time

Definition. An algorithm runs in polynomial time if its runtime (asymptotic, worst-case) is $O(\ n^k\ )$ where n is the input size and k is a constant.

Polynomial time = "good" = efficient

Jack Edmonds
(was C&O prof.)

https://en.wikipedia.org/wiki/Jack_Edmonds



1957

from his 1963 paper on a polynomial time algorithm for finding a maximum matching in a graph

2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is "good."

I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum cardinality matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

The mathematical significance of this paper rests largely on the assumption that the two preceding sentences have mathematical meaning. I am not prepared to set up the machinery necessary to give them formal meaning, nor is the present context appropriate for doing this, but I should like to explain the idea a little further informally. It may be that since one is customarily concerned with existence, convergence, finiteness, and so forth, one is not inclined to take seriously the question of the existence of a *better-than-finite* algorithm.

Polynomial time

Definition. An algorithm runs in polynomial time if its runtime (asymptotic, worst-case) is $O(n^k)$ where n is the input size and k is a constant.

Polynomial time = "good" = efficient

Is it really?

- Is $O(n^{100})$ really efficient? Well, no. Curiously, few known algorithms have runtimes like that.

- On the other hand, there are some useful algorithms that are not polynomial time (worst case):

  - simplex algorithm for linear programming
  - randomized algorithms, . . . CS 466

There are more complicated poly. time algorithms for linear programming, but it is an open question to find a pivot rule to make the simplex algorithm poly time.

W https://en.wikipedia.org/wiki/Linear_programming

Summary of Lecture 18, Part 1

We will study which problems (seemingly) cannot be solved in polynomial time.

What you should know from Lecture 18, Part 1:

- what is polynomial runtime and why it matters.

Next:

- reductions, the classes P and NP, NP-completeness

Definition. Problem X ***reduces to*** problem Y, written X $\leq$ Y, if an algorithm for Y can be used to make an algorithm for X. Think of this as "X is easier than Y".

Problem X ***reduces in polynomial time to*** problem Y, written X $\leq_P$ Y, if a polynomial time algorithm for Y can be used to make a polynomial time algorithm for X.

Note: This is actually called a "Turing reduction". Later on we will define a more specialized reduction called a "many-one reduction" to use in NP-completeness proofs.

Important consequence of X $\leq_P$ Y

If X cannot be solved in poly. time (i.e. we have a lower bound for X), then Y cannot be solved in poly. time.

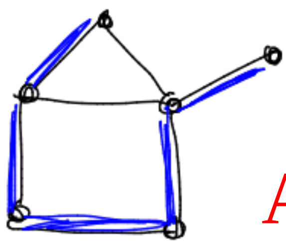
Even if we don't have an algorithm for Y or a lower bound for X, we can still use reductions to show that problems are equivalently hard (show X $\leq_P$ Y and Y $\leq_P$ X)

$\boxed{\text{we are so ignorant!}}$

$\boxed{\text{this is better than nothing}}$

Example of a reduction

A ***Hamiltonian cycle/path*** is a cycle/path that visits every vertex of a graph exactly once.

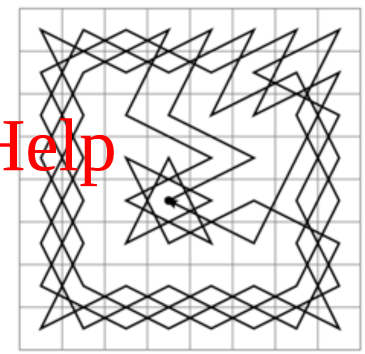


This Dalek graph has a Hamiltonian path but does not have a Hamiltonian cycle.

A Hamiltonian cycle of knight's moves on a chessboard (9th century Kashmir)

https://en.wikipedia.org/wiki/Knight%27s_tour

these links are just for enrichment, not required



Hamiltonian Cycle Problem: Given a graph, does it have a Hamiltonian cycle?
Hamiltonian Path Problem: Given a graph, does it have a Hamiltonian path?

FACT: No one knows how to solve these problems in polynomial time. The best we can do is like trying every possible vertex ordering (exponentially many).

Lemma. Hamiltonian path problem $\leq_P$ Hamiltonian cycle problem.

What this means:

If there is a poly time algorithm for Ham. cycle, then there is one for Ham. path.
If there is no poly time algorithm for Ham. path, then there is none for Ham. cycle.

Example of a reduction

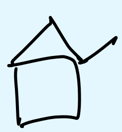
Lemma. Hamiltonian path problem $\leq_P$ Hamiltonian cycle problem.
Proof. Suppose we have a poly. time algorithm A_{cycle} for Hamiltonian cycle. We can call it like a subroutine. We want to make a poly. time algorithm for Hamiltonian path.

Input: graph G.
Output: Does G have a Hamiltonian path.

Can we just run algorithm A_{cycle} on the input G ?

A_{cycle} returns YES $\Rightarrow$ G has a Hamiltonian cycle
$\Rightarrow$ - - - path ✓

A_{cycle} returns NO $\Rightarrow$? we don't know

 no Ham. cycle but it does have Ham. path

 no Ham. cycle and no Ham. path

We need to make G' such that
G has Ham. path iff G' has Ham. cycle

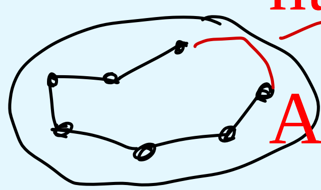
Example of a reduction

Lemma. Hamiltonian path problem $\leq_P$ Hamiltonian cycle problem.
Proof. Suppose we have a poly. time algorithm A_{cycle} for Hamiltonian cycle. We can call it like a subroutine. We want to make a poly. time algorithm for Hamiltonian path.

Input: graph G.
Output: Does G have a Hamiltonian path.

Plan:
Construct G' such that G has a Hamiltonian path iff G' has a Hamiltonian cycle.

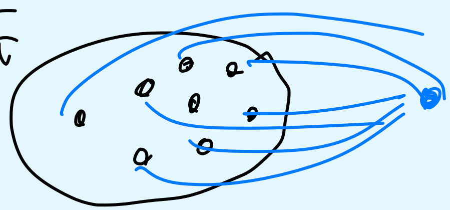
idea 1 G



add edge to get G'

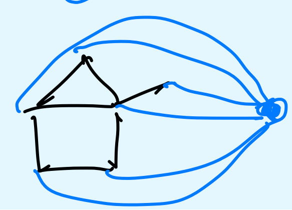
we don't know which edge.

idea 2 G



v

G' has new vertex v adjacent to all vertices of G



v

Example of a reduction

Lemma. Hamiltonian path problem $\leq_P$ Hamiltonian cycle problem.
Proof. Suppose we have a poly. time algorithm A_{cycle} for Hamiltonian cycle. We can call it like a subroutine. We want to make a poly. time algorithm for Hamiltonian path.

Algorithm A_{path}
Input: graph G
Output: Does G have a Hamiltonian path
Algorithm
1. construct graph G' by adding one new vertex v adjacent to all vertices of G.
2. run algorithm A_{cycle} on G'
3. return the YES/NO answer

As always, we must prove correctness and analyze run time.

Example of a reduction

Algorithm A_{path}
Input: graph G
Output: Does G have a Hamiltonian path
Algorithm
1. construct graph G' by adding one new vertex v adjacent to all vertices of G.
2. run algorithm A_{cycle} on G'
3. return the YES/NO answer

Run Time.

Step 1. takes linear time.

Step 2. Algorithm A_{cycle} runs in time polynomial in the size of G'.

Suppose G has n vertices and m edges. How many vertices and edges in G'?

G' has $n+1$ vertices and $m+n$ edges

Acycle runs in poly time in $2n+m+1$
That is poly. time in $n+m$

Thus the algorithm runs in polynomial time.

Example of a reduction

Algorithm A_{path}
Input: graph G
Output: Does G have a Hamiltonian path
Algorithm
1. construct graph G' by adding one new vertex v adjacent to all vertices of G.
2. run algorithm A_{cycle} on G'
3. return the YES/NO answer

Correctness

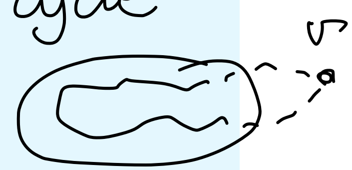
must prove two directions.

Prove: G has a Hamiltonian path iff A_{path} returns YES.

A_{path} returns YES iff A_{cycle} returns YES iff G' has Ham. cycle
by code   because A_{cycle} is correct

So prove
G has Ham. path iff G' has Ham. cycle.

$\Rightarrow$ Suppose G has Ham-path $u_1 u_2 \dots u_n$. Then G' has Ham. cycle
$v\, u_1 \dots u_n\, v$



$\Leftarrow$ Suppose G' has Ham. cycle
then delete v to get Ham. path in G

Example of a reduction

Algorithm A_{path}
Input: graph G
Output: Does G have a Hamiltonian path
Algorithm

1. construct graph G' by adding one new vertex v adjacent to all vertices of G.
2. run algorithm A_{cycle} on G'
3. return the YES/NO answer

Note the special form of the algorithm: we run algorithm A_{cycle} only once and return its answer. This is called a ***many-one reduction*** ("one-shot" reduction).

Exercise. Find a reduction in the other direction, i.e. prove:

Lemma. Hamiltonian cycle problem $\leq_P$ Hamiltonian path problem.

Given G, input to Ham. cycle
construct G' input to Ham. path
s.t. G has Ham. cycle iff G' has Ham. path

Summary of Lecture 18, Part 2

We will study which problems (seemingly) cannot be solved in polynomial time.
Reductions are a main tool to do this.

What you should know from Lecture 18, Part 2:

- what is a reduction and how to reduce one problem to another

Next:

- the classes P and NP, NP-completeness

Definition. A ***decision problem*** is one where the output is YES/NO.

The theory of NP-completeness focuses on decision problems
- it's easier that way
- optimization and decision are usually equivalent with respect to poly. time

Examples of decision problems

- Given a number, is it prime?

- Given a graph, does it have a Hamiltonian cycle?

- Given an edge-weighted graph G and a number k, does G have a TSP tour of length $\le k$?
  (TSP = Travelling Salesman Problem)

Definition. A ***decision problem*** is one where the output is YES/NO.

The theory of NP-completeness focuses on decision problems
- it's easier that way
- optimization and decision are usually equivalent with respect to poly. time

Examples of decision problems — Optimization (beyond yes/no)

- Given a number, is it prime? — Find prime factorization.

- Given a graph, does it have a Hamiltonian cycle? — Find the cycle!

- Given an edge-weighted graph G and a number k, does G have a TSP tour of length $\leq k$? — Find the tour! And find the minimum k.

  (TSP = Travelling Salesman Problem)

Equivalence of optimization and decision:

there is no general proof but things are usually ok

another open problem!

one case where they don't seem equivalent:
testing if a number is prime seems easier than finding its prime factorization (factoring)

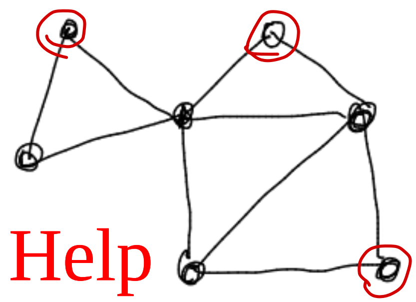
W https://en.wikipedia.org/wiki/Integer_factorization

Example — of equivalence of optimization and decision problems

(equivalence with respect to being solved in poly. time)

Maximum independent set.

Recall: an independent set in a graph is a set of vertices, no two joined by an edge.



Optimization problem: Find an independent set of maximum size

what is max ind. set?

3

Decision problem: Given k, is there an independent set of size $\geq k$

decision $\leq_P$ optimization

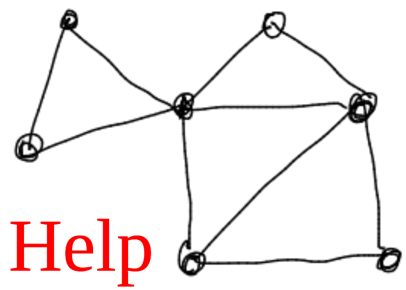
Just check if max is $\geq k$.

Example — of equivalence of optimization and decision problems

(equivalence with respect to being solved in poly. time)

Maximum independent set.

Recall: an independent set in a graph is a set of vertices, no two joined by an edge.



Optimization problem: Find an independent set of maximum size

what is max ind. set?

Decision problem: Given k, is there an independent set of size $\geq k$

optimization $\leq_P$ decision

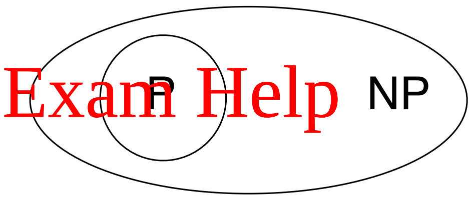
like 20 questions

Find the max k_{OPT} by testing
$k = 1, 2, \ldots, n$ using the decision alg.
Then to find an ind. set of size k_{OPT}
try deleting vertices one by one
If Max Ind. Set$(G - v) = k_{OPT}$ then $G \leftarrow G - v$
Claim. At then G is an ind. set of size k_{OPT}
Ex. prove this. Claim 2. This takes poly. time
(assuming decision alg. is poly. time)

Definition.

P = the class of decision problems that have polynomial time algorithms

What model of computing? bit complexity



The class NP — the main idea

NP is a large class of decision problems not known to be in P. The hardest problems in NP, the NP-complete problems, are all ***equivalent*** in the sense that a poly. time algorithm for one yields poly. time algorithms for all.

A few problems in NP:

- Hamiltonian path/cycle
- Travelling Salesman Problem (decision version)
- Independent Set

Common feature: if the answer is YES, then there is some succinct information (a ***certificate***) to ***verify*** that the answer is YES.

to be continued

Summary of Lecture 18, Part 3

We will study which problems (seemingly) cannot be solved in polynomial time.
We concentrate on decision problems.

What you should know from Lecture 18, Part 3:

- what is a decision problem, and why we focus on them

- how to reduce between optimization and decision problems

- the definition of the class P

Next:

- the class NP, NP-completeness