TAMPERE UNIVERSITY OF TECHNOLOGY

VII Selected Topics

~~Matrix Operations~~
~~Linear Programming~~
Number Theoretic Algorithms
~~Polynomials and the FFT~~
Approximation Algorithms
…

31 Number-Theoretic Algorithms

- Now, a “large input” typically means an input containing “large integers” rather than an input containing “many integers”
- We measure the size of an input in terms of the number of bits required to represent that input, not just the number of integers in the input
- We consider the set $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ of integers and the set $\mathbb{N} = \{0, 1, 2, \dots\}$ of natural numbers

Divisibility and divisors

- The notation $d \mid a$ ("d divides a") means that $a = kd$ for some integer k
- Every integer divides 0
- If $a > 0$ and $d \mid a$, then $|d| \leq |a|$
- If $d \mid a$, then a is a ***multiple*** of d
- If d does not divide a, we write $d \nmid a$
- If $d \mid a$ and $d \geq 0$, we say that d is a ***divisor*** of a
- $d \mid a$ if and only if $-d \mid a$, so that no generality is lost by defining the divisors to be nonnegative

Prime and composite numbers

- An integer $a > 1$ whose only divisors are the trivial divisors 1 and a is a ***prime (number)***
- An integer $a > 1$ that is not prime is a ***composite (number)***
- The integer 1 a ***unit***, and it is neither prime nor composite
- Similarly, the integer 0 and all negative integers are neither prime nor composite

$2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,\ldots$

Fermat's two square theorem

- Odd primes can be arranged in two classes
  - Those that leave remainder 1 when divided by 4
    5,13,17,29,37,41,…
  - and the primes which leave remainder 3
    3,7,11,19,23,31,…
- All primes in the 1st class, and none of the 2nd, can be expressed as a square of two integral squares

$5 = 1^2 + 2^2, 13 = 2^2 + 3^2, 17 = 1^2 + 4^2, 29 = 2^2 + 5^2, \ldots$

The division theorem, remainders, and modular equivalence

Theorem 31.1 (Division theorem)

For any integer a and any positive integer n, there exist unique integers q and r s.t. $0 \leq r < n$ and $a = qn + r$

- The value $q = \lfloor a/n \rfloor$ is the ***quotient*** of the division
- The value $r = a \bmod n$ is the ***remainder*** (or ***residue***) of the division
- We have that $n \mid a$ if and only if $a \bmod n = 0$

- We can partition the integers into n equivalence classes according to their remainders modulo n
- The equivalence class modulo n containing an integer a is

$$[a]_n = \{a + kn : k \in \mathbb{Z}\}$$

- E.g., $[3]_7 = \{\ldots, -11, -4, 3, 10, 17, \ldots\}$
- We can also denote this set by $[-4]_7$ and $[10]_7$
- We can say that writing $a \in [b]_n$ is the same as writing

$$a \equiv b \pmod{n}$$

Common divisors and greatest common divisors

- If d is a divisor of a and d is also a divisor of b, then d is a common divisor of a and b
- 1 is a common divisor of any two integers
- An important property of common divisors is that $d \mid a$ and $d \mid b$ implies $d \mid (a + b)$ and $d \mid (a - b)$
- More generally, $d \mid a$ and $d \mid b$ implies $d \mid (ax + by)$ for any integers x and y
- Also, if $a \mid b$, then either $|a| \le |b|$ or $b = 0$, which implies that $a \mid b$ and $b \mid a$ implies $a = \pm b$

- The ***greatest common divisor*** (gcd) of two integers a and b, not both zero, is the largest of the common divisors of a and b; $\gcd(a,b)$
- $\gcd(24,30)=6$, $\gcd(5,7)=1$, and $\gcd(0,9)=9$
- If a and b are both nonzero, then $\gcd(a,b)$ is an integer between 1 and $\min(|a|,|b|)$
- Define $\gcd(0,0)=0$

$$\gcd(a,b)=\gcd(b,a)$$
$$\gcd(a,b)=\gcd(-a,b)$$
$$\gcd(a,b)=\gcd(|a|,|b|)$$
$$\gcd(a,0)=|a|$$
$$\gcd(a,ka)=|a| \text{ for any } k\in\mathbb{Z}$$

Theorem 31.2 *If a and b are any integers, not both zero, then $\gcd(a,b)$ is the smallest positive element of the set $\{ax+by : x,y\in\mathbb{Z}\}$ of linear combinations of a and b.*

Corollary 31.3 *For any integers a and b, if $d \mid a$ and $d \mid b$, then $d \mid gcd(a,b)$.*

Corollary 31.4 *For all integers a and b and any nonnegative integer n, $gcd(an,bn)=n\ gcd(a,b)$.*

Corollary 31.5 *For all positive integers n, a, and b, if $n \mid ab$ and $gcd(a,n)=1$, then $n \mid b$.*

Relatively prime integers

- Two integers a and b are ***relatively prime*** if their only common divisor is 1; $\gcd(a, b) = 1$
- E.g., 8 and 15 are relatively prime, but neither is a prime number *per se*
- If two integers are each relatively prime to p, then their product is relatively prime to p

Theorem 31.6 *For any integers a, b, and p, if both $gcd(a, p) = 1$ and $gcd(b, p) = 1$, then $gcd(ab, p) = 1$.*

Proof It follows from Theorem 31.2 that there exist integers x, y, x', and y' s.t.

$$ax + py = 1$$
$$bx' + py' = 1$$

Multiplying these equations and rearranging, we have

$$ab(xx') + p(ybx' + y'ax + pyy') = 1$$

Since 1 is thus a positive linear combination of ab and p, an appeal to Theorem 31.2 completes the proof. ■

- Integers $n_1, n_2, \ldots, n_k$ are pairwise relatively prime if, whenever $i \neq j$, we have $\gcd(n_i, n_j) = 1$

Unique factorization

Theorem 31.7 *For all primes p and all integers a and b, if $p \mid ab$, then $p \mid a$ or $p \mid b$ (or both).*

Theorem 31.8 (Unique factorization) *There is exactly one way to write any composite integer a as a product of the form $a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, where the p_i are prime, $p_1 < p_2 < \cdots < p_r$, and the e_i are positive integers.*

- As an example, the number $6\,000$ is uniquely factored into primes as $2^4 \cdot 3 \cdot 5^3$

31.2 Greatest common divisor

- Prime factorizations of positive integers a and b

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$$
$$b = p_1^{f_1} p_2^{f_2} \cdots p_r^{f_r}$$

- with zero exponents being used to make the set of primes $p_1, p_2, \dots, p_r$ the same for a and b, then,

$$\gcd(a, b) = p_1^{\min(e_1, f_1)} p_2^{\min(e_2, f_2)} \cdots p_r^{\min(e_r, f_r)}$$

- However, the best algorithms to date for factoring do not run in polynomial time

- Euclid's algorithm for computing greatest common divisors relies on the following theorem

Theorem 31.9 (GCD recursion theorem)

For any integers $a \geq 0$ *and* $b > 0$,

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

- The *Elements* of Euclid (circa 300 B.C.) describes the following gcd algorithm, it may be of even earlier origin

Euclid's algorithm

EUCLID(a, b)

1. **if** $b = 0$
2. **return** a
3. **else return** EUCLID$(b, a \bmod b)$

EUCLID$(21,30)$ = EUCLID$(30,21)$
= EUCLID$(21,9)$
= EUCLID$(9,3)$
= EUCLID$(3,0)$ = 3

The running time of Euclid's algorithm

- We analyze the worst-case running time of EUCLID as a function of the size of a and b
- Assume w.l.o.g. that $a > b \geq 0$
- The overall running time of EUCLID is proportional to the number of recursive calls it makes
- Our analysis makes use of the Fibonacci numbers F_k

Lemma 31.10 *If* $a > b \geq 1$ *and the call* EUCLID(a, b) *performs* $k \geq 1$ *recursive calls, then* $a \geq F_{k+2}$ *and* $b \geq F_{k+1}$.

Proof The proof proceeds by induction on k. For the basis of the induction, let $k = 1$. Then, $b \geq 1 = F_2$, and since $a > b$, we must have $a \geq 2 = F_3$.

Since $b > (a \bmod b)$, in each recursive call the first argument is strictly larger than the second; the assumption that $a > b$ therefore holds for each recursive call.

Assume inductively that the lemma holds if $k-1$ recursive calls are made; we then prove that the lemma holds for k recursive calls. Since $k > 0$, we have $b > 0$, and EUCLID(a, b) calls EUCLID$(b, a \bmod b)$ recursively, which in turn makes $k-1$ recursive calls.

The inductive hypothesis then implies that $b \geq F_{k+1}$ (thus proving part of the lemma), and $a \bmod b \geq F_k$.

We have

$$b + (a \bmod b) = b + (a - b\lfloor a/b \rfloor) \leq a,$$

since $a > b > 0$ implies $\lfloor a/b \rfloor \leq 1$.

Thus, $a \geq b + (a \bmod b) \geq F_{k+1} + F_k = F_{k+2}$. ■

Theorem 31.11 (Lamé's theorem) *For any integer* $k \geq 1$, *if* $a > b \geq 1$ *and* $b < F_{k+1}$, *then the call* EUCLID(a, b) *makes fewer than* k *recursive calls.*

- Show (by induction on k) that the upper bound of this theorem is the best possible because the call EUCLID(F_{k+1}, F_k) makes exactly $k-1$ recursive calls when $k \geq 2$
- Since $F_k \approx \phi^k/\sqrt{5}$, where ϕ^k is the golden ratio $(1+\sqrt{5})/2$, the number of recursive calls in EUCLID is $O(\lg b)$

31.6 Powers of an element

- Just as we often consider the multiples of a given element a, modulo n, we consider the sequence of powers of a, modulo n, where $a \in \mathbb{Z}_n^*$: a^0, a^1, a^2, a^3, …. modulo n
- Indexing from 0, the 0th value in this sequence is $a^0 \bmod n = 1$, and the ith value is $a^i \bmod n$
- For example, the powers of 3 modulo 7 are

i	0	1	2	3	4	5	6	7	8	9	10	11	...
$3^i \bmod 7$	1	3	2	6	4	5	1	3	2	6	4	5	...

- Above $\mathbb{Z}_n^*$ stands for a ***multiplicative group modulo*** n: $(\mathbb{Z}_n^*, \cdot_n)$
- The elements of this group are the set $\mathbb{Z}_n^*$ of elements in $\mathbb{Z}_n$ that are relatively prime to n:

$$\mathbb{Z}_n^* = \{[a]_n \in \mathbb{Z}_n : \gcd(a, n) = 1\}$$

- An example of such a group is

$$\mathbb{Z}_{15}^* = \{1,2,4,7,8,11,13,14\}$$

- The powers of 2 modulo 7 are

i	0	1	2	3	4	5	6	7	8	9	10	11	...
2^i mod 7	1	2	4	1	2	4	1	2	4	1	2	4	...

- Let $\langle a \rangle$ denote the subgroup of $\mathbb{Z}_n^*$ generated by a by repeated multiplication, and let $\mathrm{ord}_n(a)$ (the "order of a, modulo n") denote the order of a in $\mathbb{Z}_n^*$
- E.g., $\langle 2 \rangle = \{1,2,4\}$ in $\mathbb{Z}_7^*$, and $\mathrm{ord}_7(2) = 3$
- The size of $\mathbb{Z}_n^*$ is denoted $\phi(n)$
- This function is known as Euler's phi function

- It satisfies the equation

$$\phi(n) = n \prod_{\substack{p : p \text{ is prime} \\ \text{and } p \mid n}} \left(1 - \frac{1}{p}\right)$$

so that p runs over all the primes dividing n (including n itself, if n is prime)

Theorem 31.30 (Euler's theorem)

For any integer $n > 1$,

$$a^{\phi(n)} \equiv 1 \pmod{n} \quad \text{for all } a \in \mathbb{Z}_n^*.$$

Theorem 31.31 (Fermat's little theorem)
If p is prime, then

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{for all } a \in \mathbb{Z}_p^*.$$

- Fermat's little theorem applies to every element in $\mathbb{Z}_p$ except 0, since $0 \notin \mathbb{Z}_p^*$
- For all $a \in \mathbb{Z}_p$, however, we have $a^p \equiv a \pmod{p}$ if p is prime
- E.g., $2^{7-1} = 2^6 = 64$ and $64 \bmod 7 = 1$, while $2^{6-1} = 25 = 32$ and $32 \bmod 6 = 2$:
  hence 6 is not prime
- We showed that 6 is a composite number without factoring it!

- Fermat's little theorem, thus, (almost) gives a test for primality
- We say that p passes the Fermat test at a, if
  $$a^{p-1} \equiv 1 \pmod{p}$$
- Call a number p ***pseudoprime*** if it passes Fermat tests for all smaller a relatively prime to it
- Only infrequent ***Carmichael numbers*** are pseudoprime without being prime
- If a number is not pseudoprime, it fails at least half of all Fermat tests
- We easily get a pseudoprimality algorithm with an exponentially small error probability

PSEUDOPRIME(p)

1. Select random $a_1, \dots, a_k \in \mathbb{Z}_p^*$
2. Compute a_i^{p-1} mod p for each i
3. If all values are 1 **accept**, otherwise **reject**

- If p isn't pseudoprime, it passes each randomly selected test with probability at most ½
- Probability that it passes all k tests is thus $\leq 2^{-k}$
- The algorithm operates in polynomial time
- To convert this algorithm to a primality algorithm, we should still avoid the problem with the Carmichael numbers

- A number x is a ***square root of 1***, modulo n, if it satisfies the equation $x^2 \equiv 1 \pmod{n}$
- The number 1 has exactly two square roots, 1 and -1, modulo any prime p
- For many composite numbers, including all the Carmichael numbers, 1 has 4 or more square roots
- E.g., ± 1 and ± 8 are the 4 square roots of 1 mod 21
- We can obtain square roots of 1 if p passes the Fermat test at a because
  - a^{p-1} mod $p \equiv 1$ and so
  - $a^{(p-1)/2}$ mod p is a square root of 1
- We may repeatedly divide the exponent by two, so long as the resulting exponent remains an integer

PRIME(p) % **accept** = input p is prime

1. **if** p is even, accept if $p = 2$, otherwise **reject**

2. Select random $a_1, \dots, a_k \in \mathbb{Z}_p^*$

3. **for each** $i \in \{1, \dots, k\}$

4. Compute $a^{p-1} \bmod p$, **reject** if different from 1

5. Let $p - 1 = st$ where s is odd and $t = 2^h$ is a power of 2

6. Compute the sequence $a^{s \cdot 2^0}, a^{s \cdot 2^1}, \dots, a^{s \cdot 2^h}$ modulo p

7. **if** some element of this sequence is not 1, find the last element that is not 1 and **reject** if that element is not -1

8. All test have been passed, so **accept**

Lemma *If p is an odd prime,*
$\Pr[\text{PRIME accepts } p] = 1.$

Proof If p is prime, no branch of the algorithm rejects: Rejection in line 4 means that $(a^{p-1} \bmod p) \neq 1$ and Fermat's little theorem implies that p is composite.

If rejection happens in line 7, there exists some $b \in \mathbb{Z}_p^*$ s.t.

$$b \not\equiv \pm 1 \pmod p \text{ and } b^2 \equiv 1 \pmod p.$$

Therefore $b^2 - 1 \equiv 0 \pmod p$.

Factoring yields

$$(b-1)(b+1) \equiv 0 \pmod{p},$$

which implies that $(b-1)(b+1) = cp$ for some positive integer c.

Because $b \not\equiv \pm 1 \pmod{p}$, both $b-1$ and $b+1$ are in the interval $]0, p[$.

Therefore p is composite because a multiple of a prime number cannot be expressed as a product of numbers that are smaller than it is. $\square$

- The next lemma shows that the algorithm identifies composite numbers with high probability
- An important elementary tool from number theory, *Chinese remainder theorem*, says that a one-to-one correspondence exists between $\mathbb{Z}_{pq}$ and $(\mathbb{Z}_p \times \mathbb{Z}_q)$ if p and q are relatively prime:
  - Each number $r \in \mathbb{Z}_{pq}$ corresponds to a pair (a, b), where $a \in \mathbb{Z}_p$ and $b \in \mathbb{Z}_q$ s.t.
    - $r \equiv a \pmod{p}$ and
    - $r \equiv b \pmod{q}$

Lemma *If p is an odd composite number,*

$$\Pr[\text{PRIME accepts } p] \leq 2^{-k}.$$

Proof Omitted, takes advantage of the Chinese remainder thm. □

- Let PRIMES = $\{n \mid n \text{ is a prime number in binary}\}$
- The preceding algorithm and its analysis establishes: PRIMES ∈ BPP
- Note that the probabilistic primality algorithm has *one-sided error.* When it rejects, we know that the input must be composite. An error may only occur in accepting the input.

- Thus an incorrect answer can only occur when the input is a composite number
- For all primes we get the correct answer
- The one-sided error feature is common to many probabilistic algorithms, so the special complexity class RP is designated for it:

Definition RP *is the class of languages that are recognized by probabilistic polynomial time Turing machines where inputs in the language are accepted with a probability of at least ½ and inputs not in the language are rejected with a probability of 1.*

- Our algorithm shows that COMPOSITES ∈ RP

PRIMES ∈ P

- A generalization of Fermat's little theorem:

Theorem A. *Let a and p be relatively prime and $p > 1$. p is a prime number if and only if*

$$(X - a)^p \equiv X^p - a \pmod{p}$$

- X is not important here, only the coefficients of the polynomial $(X - a)^p - (X^p - a)$ are significant
- For $0 < i < p$, the coefficient of X^i is $\binom{p}{i} a^{p-i}$
- Supposing that p is prime, $\binom{p}{i} = 0 \pmod{p}$ and hence all the coefficients are zero

- Therefore, we are left with the first term X^p and the last one $-a^p$, which is $-a$ modulo p
- Unfortunately, deciding the primality of p based on this requires an exponential time
- Agrawal (1999): it suffices to examine the polynomial $(X - a)^p$ modulo $X^r - 1$
- If r is large enough, the only composite numbers that pass the test are powers of odd primes
- On the other hand, r should be quite small so that the complexity of the approach does not grow too much
- Kayal & Saxena (2000): r doesn't have to be larger than $4(\log^2 p)$, in which case the complexity of the test procedure is only of the order $O(\log^3 n)$; i.e., belongs to P
- The result is based on an unproven claim

- A pair of odd numbers is called *Sophie Germain primes* if both q and $2q+1$ are primes (related to Fermat's last theorem)

- Agrawal, Kayal & Saxena (2002): If one can find a pair of SG primes q and $2q+1$ s.t.

$$q > 4\left(\sqrt{2q+1}\right)\cdot\log p$$

then r does not need to be larger than

$$2\left(\sqrt{2q+1}\right)\cdot\log p$$

- Unfortunately this test is recursive and has time requirement of $O(\log^{12} n)$ instead of the $O(\log^3 n)$ mentioned above

DETERMINISTIC-PRIME(p)

1. if $p = ab$ for some $b > 1$ **then reject**
2. $r \leftarrow 2$
3. while $r < p$ **do**
4. **if** $\gcd(p, r) \neq 1$ **then reject**
5. **if** DETERMINISTIC-PRIME(r) **then** % $r > 2$
6. Let q be the largest factor of $r-1$
7. **if** $q > 4\text{sqrt}(r)\cdot\log p$ **and** $p^{(r-1)/q} \neq 1 \pmod r$ **then break**
8. $r \leftarrow r+1$
9. for $a \leftarrow 1$ **to** $2\text{sqrt}(r)\cdot\log p$ **do**
10. **if** $(x-a)^p \neq x^p - a \pmod{x^r - 1, p}$ **then reject**
11. accept the input;

- The test of line 1 removes the powers of odd primes as required by the test of Agrawal (1999)
- The loop in lines 3–8 searches a pair of Sophie Germain primes q and r
- Line 4 tests for Theorem A that p and r are relatively prime
- The loop in line 9 examines primality using a variation of Theorem A (Agrawal, 1999) up to value $2\sqrt{r}\log p$ (AKS, 2002)
- Because Theorem A holds if and only if p is prime, the decision of the algorithm is correct
- The other variations only affect the complexity of the algorithm, not its correctness

35 Approximation Algorithms

- Many problems of practical significance are NP-complete, yet too important to abandon
- We have ways to get around NP-completeness

1) If the actual inputs are small, an algorithm with exponential running time may be satisfactory
2) We may be able to isolate important special cases that we can solve in polynomial time
3) We might come up with approaches to find near-optimal solutions in polynomial time. In practice, near-optimality is often good enough. Such an algorithm is called an ***approximation algorithm***

Performance ratios for approximation algorithms

- Let each potential solution have a positive cost; we wish to find a near-optimal solution
- The problem may be either a maximization or a minimization problem
- An algorithm has approximation ratio of $\rho(n)$ if, for any input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

- An algorithm that achieves an approximation ratio $\rho(n)$, is a $\rho(n)$-*approximation algorithm*
- The definitions apply to both minimization and maximization problems
- For a maximization problem, $0 < C \leq C^*$, and the ratio C^*/C gives the factor by which the cost of an optimal solution is larger than the cost of the approximate solution
- Similarly, for a minimization problem, $0 < C^* \leq C$, and the ratio C/C^* gives the factor by which the cost of the approximate solution is larger than the cost of an optimal solution

- We assume that all solutions have positive cost, these ratios are always well defined
- The approximation ratio of an approximation algorithm is never less than 1, since $C/C^* \leq 1$ implies $C^*/C \geq 1$
- A 1-approximation algorithm produces an optimal solution
- An approximation algorithm with a large approximation ratio may return a solution that is much worse than optimal

- An ***approximation scheme*** for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\epsilon > 0$ such that for any fixed ϵ, the scheme is a $(1+\epsilon)$-approximation algorithm
- An approximation scheme is a polynomial-time approximation scheme if for any fixed $\epsilon > 0$, it runs in time polynomial in the size n of its input
- The running time of a poly-time approximation scheme can increase rapidly as ϵ decreases
- E.g., the running time of a polynomial-time approximation scheme might be $O(n^{n/2})$

- Ideally, if ϵ decreases by a constant factor, the running time to achieve the desired approximation should not increase by more than a constant factor (though not necessarily the same constant factor by which ϵ decreased)
- The running time of a ***fully polynomial-time approximation scheme*** is polynomial in both $1/\epsilon$ and the size n of the input instance
- E.g., the running time might be $O((1/\epsilon)^2 n^3)$
- With such a scheme, any constant-factor decrease in ϵ comes with a corresponding constant-factor increase in the running time

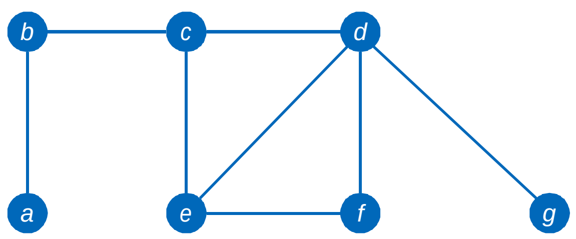
35.1 The vertex-cover problem

- A **vertex cover** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ s.t. if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both)
- The size of a vertex cover is the number of vertices in it
- The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given graph
- This problem is the optimization version of an NP-complete decision problem

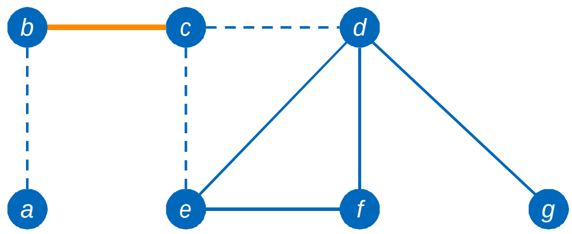
APPROX-VERTEX-COVER(G)

1. $C \leftarrow \emptyset$
2. $E' \leftarrow G.E$
3. **while** $E' \neq \emptyset$
4. let (u, v) be an arbitrary edge of E'
5. $C \leftarrow C \cup \{u, v\}$
6. remove from E' every edge incident on either u or v
7. **return** C

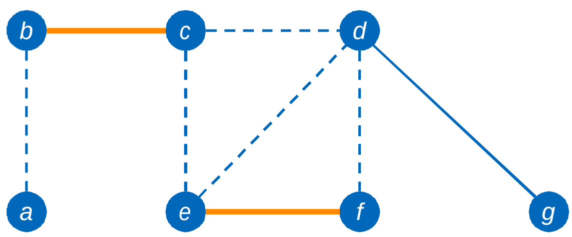
Selection of the first random edge: (b, c)



We remove other edges connected with nodes *b* and *c*

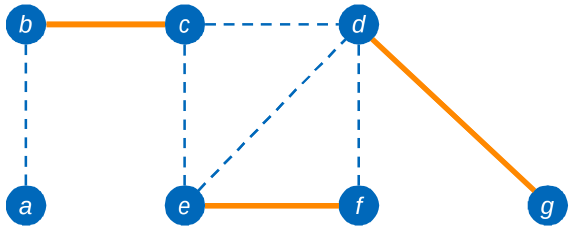


The next random choice : (*e*, *f*) and
Removal of other edges connected with its nodes



The only remaining choice (d, g)

We end up with a cover of 6 nodes,
while the optimal one has 3 nodes (e.g., b, d, e)



Theorem 35.1 APPROX-VERTEX-COVER *is polynomial time 2-approximation algorithm for vertex cover.*

Proof. The time complexity of the algorithm, using adjacency list representation for the graph, is $O(V + E)$, and thus uses a polynomial time.

The set of nodes C returned by the algorithm obviously is a vertex cover for the edges of G, because nodes are inserted into C in the loop of row 3 until all edges have been covered.

- Let A be the set of edges chosen by algorithm in row 4.
- In order to cover the edges of A any vertex cover — in particular also the optimal vertex cover C^*— has to contain at least one of the ends of each edge in A.
- Because the end points of the edges in A are distinct by the design of the algorithm, $|A|$ is a lower bound for the size of any vertex cover.
- In particular,

$$|C^*| \geq |A|.$$

- Each execution of line 4 picks an edge for which neither of its endpoints is already in C, yielding an (exact) upper bound on the size of the vertex cover returned:

$$|C| = 2|A|$$

- Combining the above equations, we obtain

$$|C| = 2|A| \leq 2|C^*|$$

- thereby proving the theorem. ■