ICS 353 Final Notes

Chapter 1 — Basic Concepts in Algorithmic Analysis

- searching
  - linear search
    - ```
      Algorithm 1.1 LINEARSEARCH
      Input: An array A[1..n]
      Output: j if x = A[j], else 0
      j <- 1
      while (j < n) and (x != A[j])
         j <- j + 1
      end while
      if x = A[j] then return j else return 0
      ```
    - O(n)
  - binary search
    - check middle
      - if key less, check first half
      - if more, check second half
      - repeat
    - ```
      Algorithm 1.2 BINARYSEARCH
      Input: A[1..n]
      Output: j, 0

      low <- 1; h <- n; j <- 0
      while (low <= high) and (j = )
         mid <- ⌊(low + high)/2⌋
         if x = A[mid] then j <- mid
         else if x < A[mid] then high <- mid - 1
         else low <- mid + 1
      end while
      return j
      ```
- merge
  - combine two (sorted) arrays
  - iterate over both arrays
  - whenever u get the smallest of the two, insert it into the new array
    - array with smallest's pointer gets increment

        - e.g. increment s when smallest element from array 1 added to new array
    - if one of the arrays is exhausted (shorter than the other), append the remaining elements of the other array into the new array
    - number of element comparisons
        - best case
            - smaller of two arrays
            - n1
        - worst case
            - n - 1 where n1 + n2 = n
    - ```
      Algorithm 1.3 MERGE
      Input: A[1..m], pointers p, q, and r
      Output: A[p..r] (merged array)

      s <- p; t <- q + 1; k <- p
      while s <= q and t <= r
         if A[s] <= A[t] then
            B[k] <- A[s]
            s <- s + 1
         else
            B[k] <- A[t]
            t <- t + 1
         end if
         k <- k + 1
      end while
      if s = q + 1 then B[k..r] <- A[t..r]
      else B[k..r] <- A[s..q]
      ed if
      A[p..r] <- B[p..r]
      ```
- sorting
    - selection sort
        - find the minimum and store in A[1]
            - then the next minimum in A[2]
                - etc.
        - number of element comparisons
            - (n-1) + n(- 2) + ... + 1 = n(n-1) / 2
      ```

- ```
  Algorithm 1.4 SELECTIONSORT
  Input: A[1..n]
  Output: A[1..n] sorted in nondecreasing order
  for i <- 1 to n - 1
     k <- i
     for j <- i + 1 to n
        if A[j] < A[k] then k <- j
     end for
     if k != i then interchange A[i] and A[k]
  end for
  ```

- insertion sort
  - ```
    Algorithm 1.5 INSERTIONSORT
    Input: A[1..n]
    Output: A[1..n] sorted in nondecreasing order

    for i <- 2 to n
       x <- A[i]
       j <- i - 1
       while (j > 0) and (A[j] > x)
          A[j + 1] <- A[j]
          j <- j - 1
       end while
       A[j + 1] <- x
    end for
    ```
  - number of element comparisons
    - best case (already sorted)
      - n - 1
    - worst case (sorted in decreasing order)
      - n(n-1) / 2
- bottom-up merge sort
  - divide element into pairs
    - merge each pair into 2 element sequences
      - merge further in 4 element sequences
        - etc.
  - number of element comparisons
    - best case
      - nlogn / 2
    - worst case
      - nlogn - n + 1

- time complexity
  - elementary operations
    - arithmetic
    - comparison and logical
    - assignments
  - O-notation
    - f(n) is O(g(n)) if there exists a number n0 and constant c > 0 such that
      - $\forall n \leq n_0, f(n) \leq cg(n)$
  - $\Omega - notation$
    - $\forall n \leq n_0, f(n) \geq cg(n)$
  - $\Theta - notation$
    - $\forall n \geq n_0, c_1 g(n) \leq f(n) lec_2 g(n)$
- space complexity
  - linear search, binary search, selection sort, insertion sort
    - O(1)
  - merge
    - O(n)
  - bottom up sort
    - O(n)
- worst case analysis
  - select the maximum cost among all possible iputs of size n
- average case analysis
  - proabilities of all inputs of size n
  - multiply by cost for each
- how to solve recurrence relations
    - expansion
    - substitution
    - change of variables

Chapter 4 — Induction

- finding the majority element
  - set a counter to 1 and let x = A[1]
    - increase by 1 if element is equal to x
    - decrease by 1 if not
  - return x as candidate if counter is more than 1
  - if counter becomes zero when comparing then call procedure recurisvely on the next element
  - running time
    - O(n)
- integer exponentiation
  - O(logn)
    - linear in the input size
  - if n is even then x^n = (x^m)^2
    - if odd, x^n = x(x^m)^2
  - recursive exp
    - power(x, m):
      - if m = 0 then y ← 1
      - y ← power(x, m/2)
      - square y
      - m is odd then y ← xy
      - return y
  - iterative exp
    - to compute x^n
    - y ← 1
    - for each bit in the power n
      - y ← y^2
      - if the bit is 1 then y ← xy
    - return y
- horner's algorithm
  - $$P_n(x) = xP_{n-1}(x) + a_0$$
- radix sort
  - sort by most significant digit first
  - O(kn)
    - k is the length of the longest digit

    - O(n) space

Chapter 5 — Divide and Conquer

- binary search
    - recursive
- mergesort
    - if low < high
        - get mid
        - call mergesort from low to mid
        - call mergesort from mid + 1 to high
        - merge both results
- divide and conquer paradigm
    - divide
    - conquer
    - combine
- select
    - to find kth smallest element
    - divide A into q groups of 5
        - discard remaining elements if doesnt divide p
    - sort each individually and get the median
        - let the set of medians be M
    - mm ← select(M, q/2)
    - parittion A into 3
        - A1 where a < mm
        - A2 where a = mm
        - A3 where a > mm
    - if |A1| >= k return select(A1, k)
    - elif |A1| + |A2| >= k return mm
    - elif |A1| + |A2| < k: return select(A3, k - |A1| - |A2|)
    - O(n)
        - e.g. median in O(n)
    - T(n/5) to find median of medians
    - T(3n/4) to recursively call select on on A1 or A3

    - cn to parittion and sort groups of 5
- quicksort
    - in place sorting
    - partitioning
        - split algorithm
    - sorting algorithm
        - if low < high
            - SPLIT(A, w) // w is the new pos of A[low]
            - quicksort(A, low, w - 1)
            - quicksort(A, w + 1, high)
    - complexity
        - worst case
            - O(n^2)
        - average case
            - O(nlogn)
        - when median chosen as pivot
            - O(nlogn)
- integer multiplication
    - naive algorithm
        - O(n^2)
    - each integer is 2 parts of n/2 bits each
    - $$uv = (w2^{n/2} + x)(y2^{n/2} + z)$$
    - can be simplified to:
    - $$wy2^n + (wz + xy)2^{n/2} + xz$$
        - multiply by 2^n means shifting by n bits
            - O(n) time
        - 4 multiplications and 3 additions
        - O(n^2)
            - still not an improvement over naive algorithm
    - if you compute wz + xy as (w+x)(y+z) - wy - xz
        - then its 3 multiplications and 6 additions/substractions
        - O(n^log3)
- matrix multiplication

- o(n^3) time complexity
  - 7 multiplications and 18 additions of n/2*n/2 matrices
- strassen's algorithm
  - O(n^log7)
    - more additions
    - but much less multiplications

Chapter 6 — Dynamic Programming

- longest common subsequence
  - L[i, j] = { 0 if i = 0 or j = 0
    - L[i - 1, j -1] + 1 if ai = bj
    - max{L[i, j - 1], L[i - 1, j] ai != bh
  - O(nm)
  - O(min{m, n}) space
- matrix chain multiplication
  - do each diagonal
  - O(n^3)
  - O(n^2) space

$$C[i,j] = \min_{i<k\leq j} \{C[i,k-1] + C[k,j] + r_i r_k r_{j+1}\}.$$

- all pairs shortest path
  - O(n^3)
  - space O(n^2)
  - $d_{i,j}^k = \{l[i,j] \quad if\, k = 0, min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\}$
- knapsack
  - V[i, j] = { 0
    - V[i - 1, j] if j < si
    - max{V[i - 1, j], V[i - 1, j - si] + vi} if j >= si
  - O(nC)
  - O(C) space

Parallel Algorithms

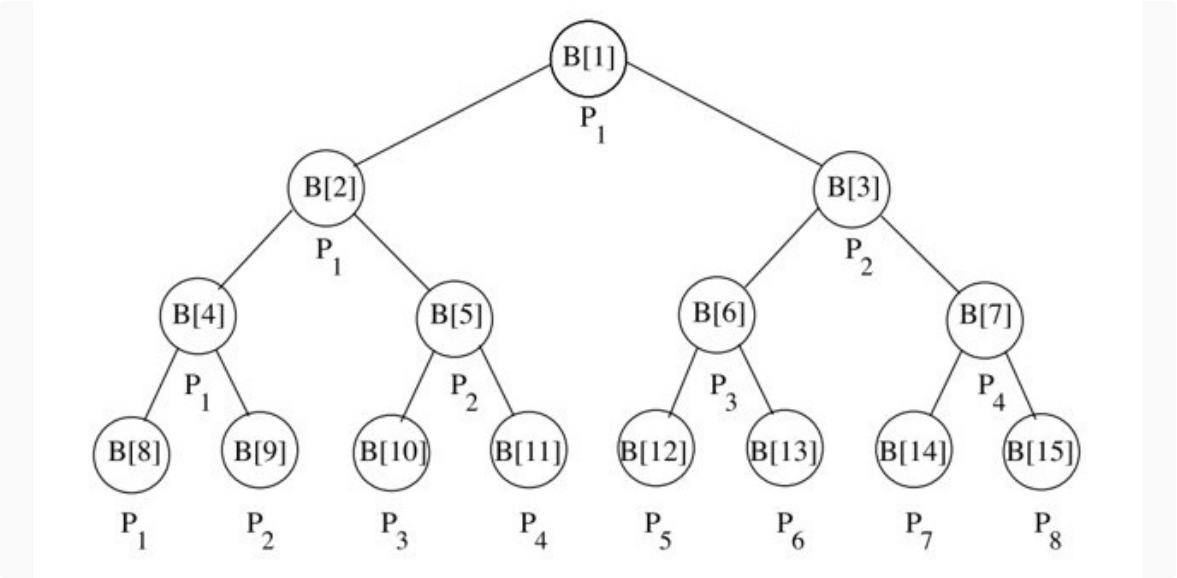
- parallel computing
  - architecture where several processors execute or process an application simultaneously
- parallel algorithm
  - algorithm that makes use of more than one processor to solve a specific problem
- adding n numbers
  - sequential solution
    - iterative
    - O(n)
  - parallel solution
    - add pairs
    - then add sums of pairs
    - log n times
    - n/2 processors needed
- find the index of a key
  - sequential solution
    - go through each element and return the index if matched
    - O(n)
  - parallel solution
    - every element in a processor
    - processor P1 sets j to 0 O(1)
    - each processor compares their element to the key O(1)
    - if found, one processor writes on j O(1)
    - O(1) total
- performance of parallel algorithms
  - T(n, p)
    - running time using p processors
  - C(n, p)
    - cost of the algorithm
      - running time * number of processors
      - pT(n, p)
  - W(n, p)
    - total number of operations done by individual processors
  - S(p)

        - speedup
        - T(n, 1) / T(n, p)
        - if S(p) = p
            - speedup is perfect
    - E(n, p)
        - efficiency
        - S(p) / p
        - T(n, 1) / C(n, p)
- parallel architectures
    - SISD
        - single instruction stream, single data stream
        - one cpu
    - SIMD
        - single program & multiple CPUs
    - MISD
    - MIMD
        - p processors, p streams of instructions, and p streams of data
        - most parallel computers
- shared-memory computers
    - RAM
        - for sequential
    - PRAM
        - parallel
        - has no limit on the number of processors
- interconnection-network computers
    - processor-memory pairs connected in a pattern
        - G = (V, E)
        - V is the set of processors
        - E is the set of two-way links
    - degree of a network
        - maximum degree of any vertex in the graph
    - network diameter
        - maximum shortest path distance between any two processors
    - bisection width

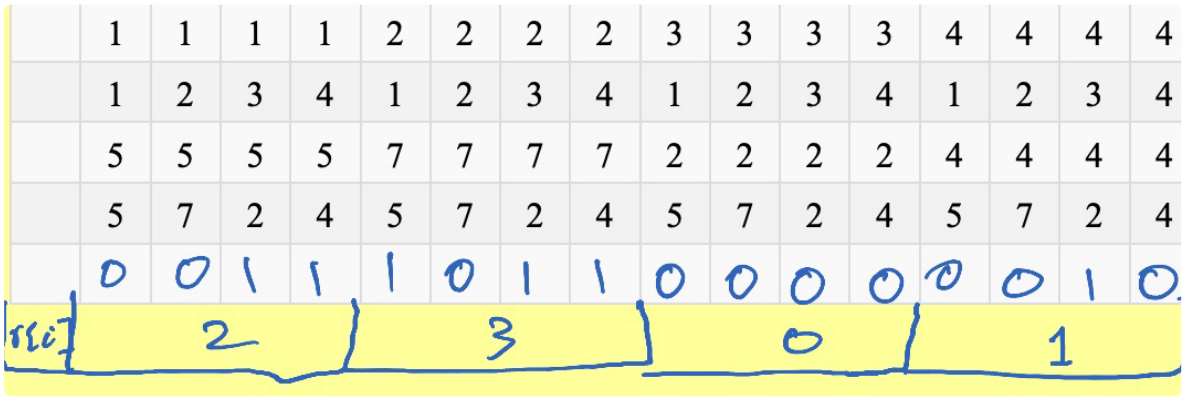
        - minimum number of links that have to be removed in order to disconnect the network into two equal size subnetworks

Shared Memory Computers (PRAM)

- read/write conflicts are resolved by
    - EREW
        - exclusive read exclusive write
    - CREW
        - concurrent read exclusive write
    - ERCW
    - CRCW
        - writes are defined as
            - COMMON
                - successful if all processors write the same value
            - ARBITRARY
                - only one arbitrary attempt is successful
            - PRIORITY
                - processors are ranked
                - highest rank can write
            - Array reduction
                - e.g. SUM, AND, MAX
- balanced tree method
    - to add n numbers
        - n = 2^k
    - each leaf node is assigned processor Pi
        - each input in position b[n] to b[2n - 1]



    - each internal node at level j (0 to k - 1) is assigned processor P1 to P_2^j
        - e.g. level 2 has processors from 1 to 2^2 = 4 (1, 2, 3, 4)
- brent theorem
    - number of procsesors can be reduced without affecting the time complexity
        - let the number of processors be n / log n
            - assign log n numbers to each processor
        - each processor finds the maximum in its group, sequentially, using O(lgn) comparisons
        - the algorithm continues to find the maximum of the n/logn maxima in O(lgn)
    - therefore the running time is O(lgn) and the cost is O(n)
    - formally
        - algorithm requires tp parallel steps using p processors
        - total number of operations is s
        - let q = s / tp
        - there exists an algorithm that performs at most 2tp steps using q processors
        - if sequential time complexity is O(s), then Aq is optimal
- sorting
    - CRCW
        - SUM
        - n^2 processors
        - if A[i] > A[j] then r[i] ← 1 else 0
        - running time is O(n)
        - cost is O(n^2)

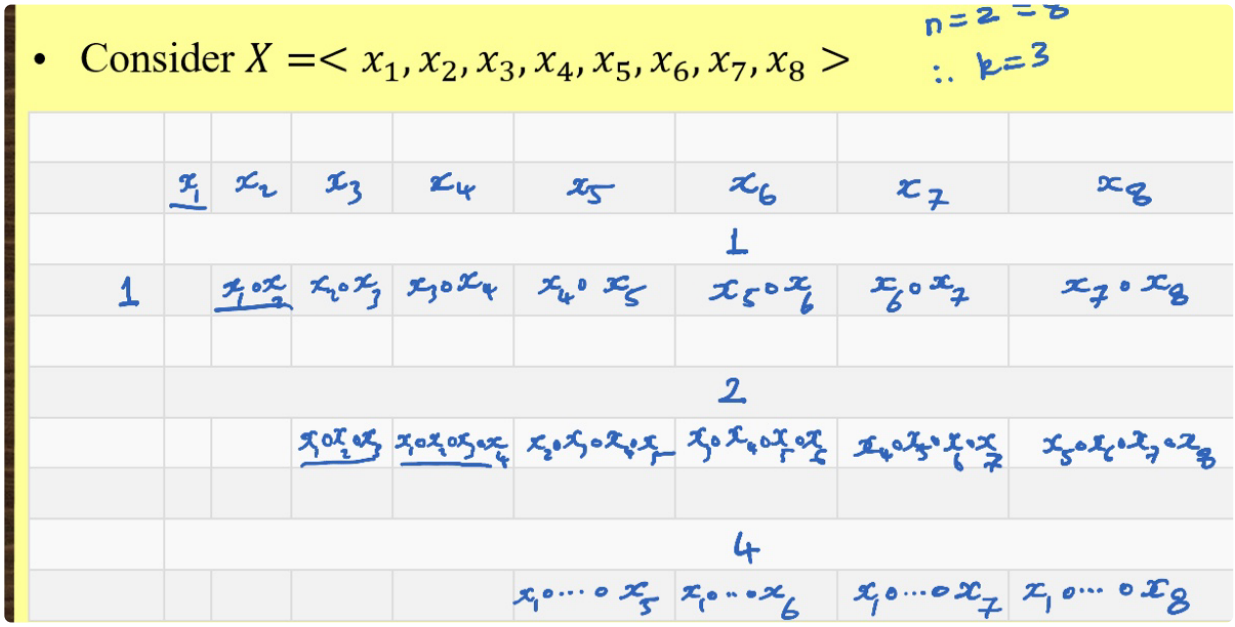


- EREW
  - use n processors to initialize r[1..n]
  - use n processors to copy array to C
  - sequentially from i=1 to n - 1
    - in parallel from j=1 to n
      - k ← sum of i and j wrapped around
      - if A[j] > C[k] then increase rank of r[j]
  - running time is O(n)
  - cost is O(n^2)
- parallel prefix
  - O(lgn)
  - cost: O(nlogn)

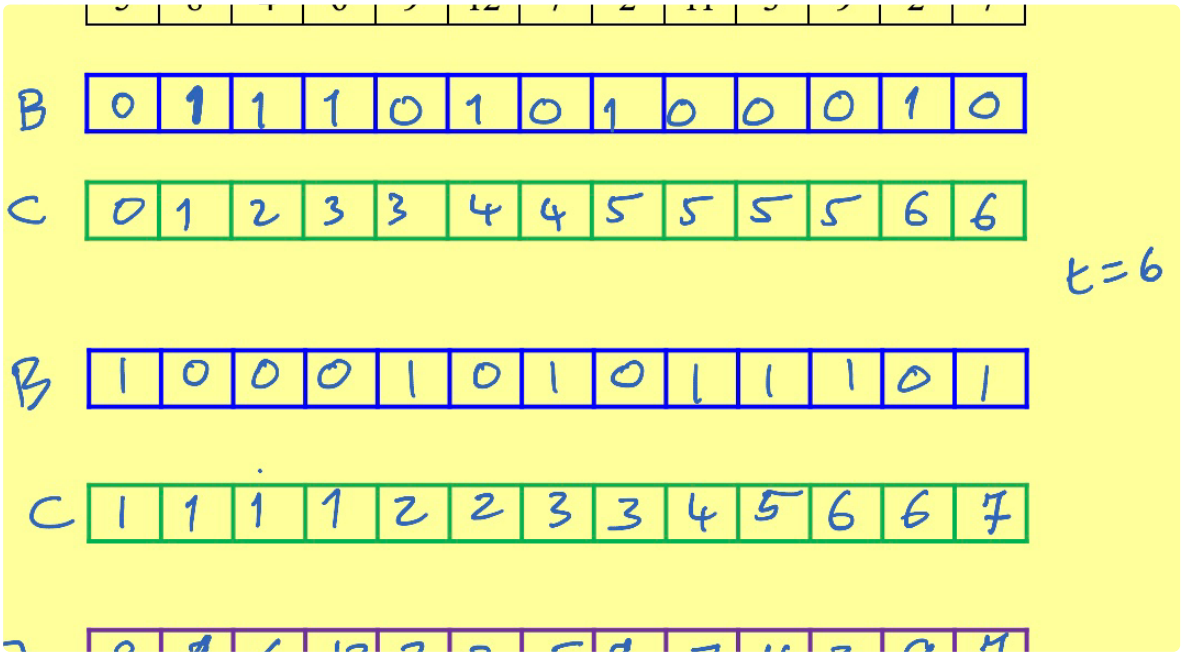
Algorithm 2.5 PARPREFIX
Input: $X = \langle x_1, x_2, \ldots, x_n \rangle$, a sequences of n numbers, where $n = 2^k$.
Output: $S = \langle s_1, s_2, \ldots, s_n \rangle$, the prefix sums of X.

```
1.  for i ← 1 to n do in parallel
2.      s_i ← x_i
3.  end for
4.  t ← 1
5.  for j ← 1 to k do
6.      for i ← t + 1 to n do in parallel
7.          s_i ← s_{i−t} ∘ s_i
8.      end for
9.      t ← 2t
10. end for
11. return S
```

- Consider $X = < x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 >$ $n = 2^3 = 8$ $\therefore k = 3$



- array packing
  - let A be an array of n elements such that t of them are marked
  - create an array D such that all marked elements appear before all unmarked elements (preserving the order)
  - algorithm:
    - create an array B assigning 1 to each marked element
    - apply the parallel prefix algorithm to B and store sums in C
    - if aj is marked, store it in array D
    - exchange the bits of array B
    - apply the parallel prefix algorithm again and store in C
    - if aj is unmarked, store it in array D in position t + ci



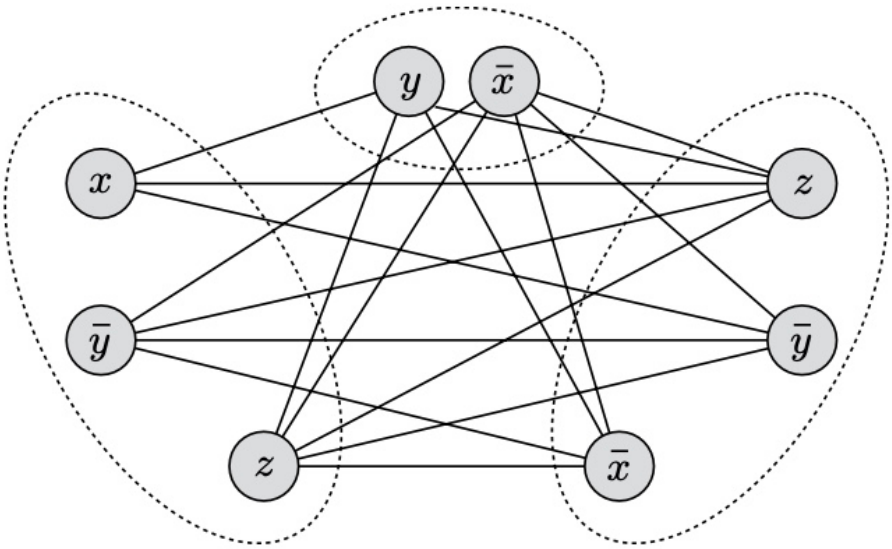
- O(logn)
- cost: O(nlogn)

Chapter 9 — NP-complete Problems

- class P
  - consists of decision problems whose yes/no solution can be obtained using a deterministic algorithm that runs in polynomial time
- class NP
  - problems for which there exists a deterministic algorithm which can verify the correctness of a solution in polynomial time
  - nondeterministic algorithm
    - guessing phase
      - arbitrary string of characters generated
      - polynomial time
    - verificiation phase
      - deterministic algorithm verifies two things
        - checks whether solution is in the proper format
        - checks whether it is a solution
        - answers yes or no
        - polynomial time

    - class of problems for which there exists a nondeterministic algorithm that runs in polynomial time
- NP-complete problems
    - subclass of decision problems in NP that are hardest in the sense that if one is proved to be solvable in polynomial time deterministic then all problems in NP are solvable by a polynomial time deterministic algorithm (i.e. NP = P)
    - pi reduces to pi′ in polynomial time
        - there exists a deterministic algorithm that transforms an instance I of problem pi to an instance I′ of problem pi′ in polynomial time
    - NP-hard
        - for every problem pi′ in NP, pi′ reduces to pi
    - NP + NP-hard = NP-complete
- proving np completeness
    - prove pi is in NP
    - prove that there is a NP complete problem pi′ that reduces to pi
        - i.e. prove its NP hard
- hamiltonian cycle reduces in polynomial time to traveling salesman
    - make each edge that exists in the cycle have weight 1
    - each edge that doesnt have weight n
    - let k = n
    - G has a hamiltonian cycle if and only if G′ has a tour of length exactly n
- satisfiability reduces to clique
    - construct a graph G where the vertices are all the occurrences of the 2n literals (literal and its negation)
    - make an edge if xi and xj are in different clauses and they are not negations of each other

$$f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z).$$



- satisfiability reduces to vertex cover
  - for each variable make a pair of vertices connecting xi and xi′
  - for each clause, G contains a clique Cj of size nj
  - for each vertex in Cj, there is an edge connecting w to its corresponding literal in the vertex pairs (xi, xi′)
  - k = n + sum from j=1 to m for (nj - 1)

$$f = (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y).$$

It should be emphasized that the instance I' is not only the figure shown; it also includes the integer $k = 3 + 2 + 1 = 6$. A boolean assignment of $x = true, y = true$, and $z = false$ satisfies f. This assignment corresponds to the six covering vertices shown shaded in the figure.

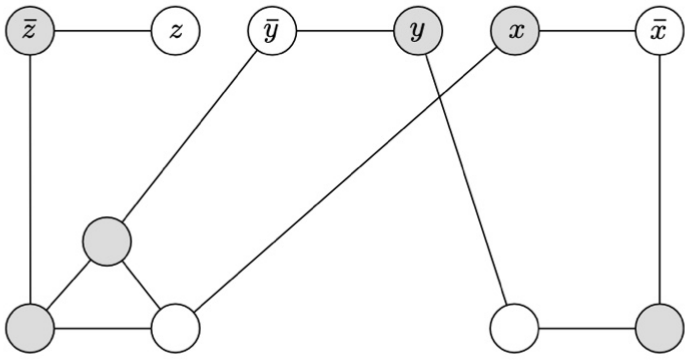


Fig. 9.2 Reducing SATISFIABILITY to VERTEX COVER

- vertex cover reduces to independent set
  - Let G be a connected undirected graph
  - S is an independent set if and only if V - S is a vertex cover in G

- vertex cover, independent set, and clique are NP complete

Basic Computability

- turing machine
  - infinite length tape
  - inptu is provided as a finite sequence of symbols
  - head reads the input tape
    - optionally replaces the symbol with another
    - changes its internal state
    - moves one cell right or left
  - starts at state s
- standard Turing machine
  - six-tuple M = (Γ, B, Q, δ, s, h)
    - Γ: tape alphabet without B
    - B: blank symbol
    - Q: set of states
    - δ: next-state function
      - δ: (Q x (Γ ∪ {B})) → (Q ∪ {h}) x (Γ ∪ B) x {L, R}
      - {L, R} is left right
    - s: initial state
    - h: accepting halt state
      - not a member of Q
      - TM cannot exit from h
- TM M accepts an input string w ∈ Γ*
  - if when started in state s with w left-adjusted on its blank tape, the last state entered is h
- TM M accepts the language L(M) consisting of all strings accepted by M
- if a TM M halts on all inputs, M recognizes the language that it accepts
  - Language L is decidable or recursive
- languages accepted by M are called recursively enumerable
- three possibilities on given input w:
  1. TM M eventually enters h (accepting)
     - w ∈ L(M)

2. TM M eventually enters h (rejecting) or crashes
    - $w \notin L(M)$
3. M never halts
    - $w \notin L(M)$

- recognizer
    - TM recognizes L(M)
- decider
    - TM recognizes L(M) and never enters an infinite loop
- decidable vs recognizable
    - decidable language is always a recognizable language
    - recognizable language:
        - language for which there is at least one machine that has it as its language
    - decidable language:
        - if there is a machine that recognizes L and that whatever the input you give it, will always either accept / reject, the language is considered decidable.
- a language $L \subseteq \Gamma^*$
    - is in P if there is a TM M with tape alphabet Γ and a polynomial p(n) such that for every $w \in \Gamma^*$, M halts in p(|w|) steps and accepts w if it is in L and rejects it otherwise
- nondeterministic Turing machine
    - seven-tuple $M = (\Sigma, \Gamma, B, Q, \delta, s, h)$
        - Σ: choice input alphabet
        - δ: next-state function
            - $\delta: Q \times \Sigma \times (\Gamma \cup \{B\}) \rightarrow (Q \cup \{h\}) \times (\Gamma \cup B) \times \{L, R\} \cup \perp$
            - if $\delta(q,c,a) = \perp$, then there is no sucessor to the current state
            - If $\delta(q,c,a) = (q', a', C)$
                - M's control unit enters state q', writes a' in the cell under the head, and moves the head left or right depending on C
    - accepts the input string if there is some choice input string such that the last state entered by M is h
    - language is in NP if there is a nondeterministic Turing machine M and polynomial (pn) such that M accepts L and for each w in L, there is a choice input c such that M halts in p(|w|) steps
- k-tape turing machine
    - for each k-tape TM,
        - there is a one tape TM such that the k-tape TM can be simulated in O(T^2) steps
- if a nondeterministic TM has more than two nondeterministic choices for a particular state and letter

    - we can design another NDTM that has at most two choices
- any language accepted by a NDTM can be accepted by a TM
- any language accepted by a nondeterministic multi-tape TM can be accepted by a deterministic TM
- universal Turing machine
    - turning machine that can simulate the behavior of an arbitrary Turing machine
    - computer
    - MUniversal
        - inputs (M, I)
            - M: turing machine description
            - I: the input to the machine M
        - results:
            - L(MUniversal) = {(M, I) | M accepts I}
- language that is recursively enumarable
    - a language where a TM that accepts it exists
- unsolvable language
    - no TM exists that accepts the language
- consider L = {w | w not accepted by M}
    - L is not recursively enumerable
    - the complement of L1 is a recursively enumerable language that is not decidable
        - complement of a decidable language is decidable
-

ignore this vvvvvv

Chapter 10 — Introduction to Computation Complexity

- alphabet
    - Σ
    - finite set of symbols
- language

    - subset of the set of all finite length strings of symbols chosen from Σ
    - Σ*
- standard Turing machine
    - one worktape
        - divided into separate cells
    - ability to read and rewrite the symbol contained in the cell of teh worktape currently scanned
- k-Tape Turing machine
    - k worktapes and k worktape heads
    - M = (S, Σ, Γ, δ, p0, pf)
        - S is the finite set of states
        - Γ is the finite set of tape symbols which include the symbol B (blank symbol)
        - Σ ⊆ Γ - {B}, the set of input symbols
        - δ is the transition function
            - function that maps elements of S * Γ^k into finite subsets of S x ((Γ - {B}) * {L, P, R})^k
        - p0 is the initial state
        - pf is the final or accepting state
    - deterministic if for every p ∈ S

Chapter 12 — Backtracking

- for some problems there does not exist an algorithm other than exhaustive search
    - need arises for developing techniques of searching
        - cut down the search space
- backtracking
    - suitable for solving problems where a large but finite number of solutions have to be inspected
-