DESIGN & ANALYSIS of ALGORITHMS

unit - 5

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

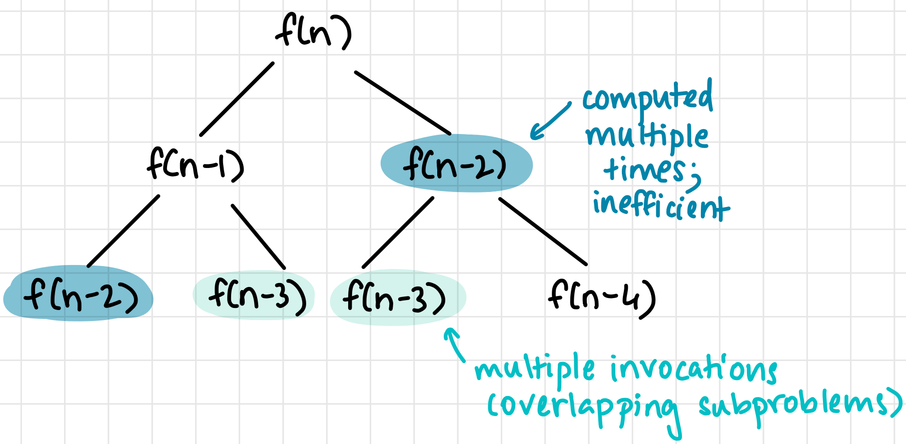
DYNAMIC PROGRAMMING

- Richard Bellman in 1950s
- Recurrence relation between larger and smaller solutions, solve smaller instances
- Record solutions in a table
- Prevents duplication of effort (subproblem) using a table and bottom-up approach

1. FIBONACCI NUMBERS

$$f(n) = f(n-1) + f(n-2)$$
$$f(0) = 0$$
$$f(1) = 1$$

Recursion tree



eg: $f(0) = 0$
$f(1) = 1$
$f(2) = 0 + 1 = 1$
$f(3) = 1 + 1 = 2$
$f(4) = 2 + 1$
$\vdots$

constant amount of work at every step

Complexity

- time: $\Theta(n)$
- space: $\Theta(n)$ if all kept
  space: $\Theta(1)$ if only prev 2 entries

2. BINOMIAL COEFFICIENT

$$(a+b)^n = C(n,0)\, a^n b^0 + \dots + C(n,k)\, a^{n-k} b^k + \dots + C(n,n)\, a^0 b^0$$

- Given n & k, compute ${}^nC_k$

Recurrence

$$C(n,k) = C(n-1,k) + C(n-1,k-1) \quad \text{for } n > k > 0$$

$$C(n,0) = 1 \quad \text{for } n \geq 0$$
$$C(n,n) = 1$$

Table

	0	1	2	3	...	k-1	k
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
⋮	⋮						
n-1	1	n-1	...			C(n-1,k-1)	C(n-1, k)
n	1	n	...				C(n,k)

pascal's triangle

$C(n,0) = 1$
$C(n,n) = 1$
$C(n,k) = C(n-1,k) + C(n-1,k-1)$

$C(2,1) = C(1,1) + C(1,0)$

Algorithm C(n,k)

```
// input: integers n ≥ 0, k ≥ 0
// output: C(n,k)

for i=0 to n
    for j=0 to min(i,k)

        if j=0 or j=i
            C[i,j] =1
        else
            C[i,j] = C[i-1,j] + C[i-1,j-1]

return C[n,k]
```

Complexity

- Time: $\Theta(nk)$
- Space: $\Theta(nk)$

Q: What does DP have in common with divide and conquer? What is the principal difference between them?

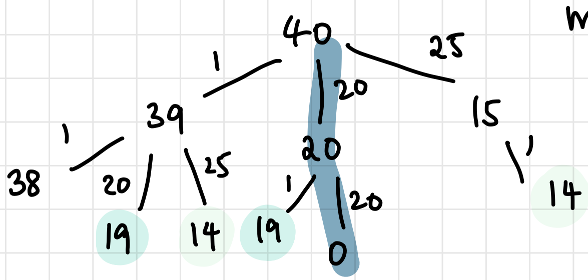
- recursive trees
- algorithm same, only computations reduced due to storing of values

Q: The coin change problem does not have an optimal greedy solution in all cases

eg: coins 1, 20, 25 and amount 40

Is there a DP based algorithm that can solve all cases of the coin change problem?

Brute force:



maintain DP array?

https://trykv.medium.com/how-to-solve-minimum-coin-change-f96a758ccade

3. KNAPSACK PROBLEM

- bag with capacity M, objects with weights and values
- 0/1 knapsack; object either picked up or not (no fractions)
- Optimisation: maximise profit due to objects; find most valuable subset of items
- eg: a thief tries to maximise profit with finite bag size (weight and value)
- exhaustive search (all subsets found, value and weight calculated, optimised subset found)
- 2^n subsets

DP Algorithm

- Derive recurrence relation that expresses a solution to an instance of the knapsack problem in terms of solutions to its smaller subinstances
- Consider knapsack $(\underbrace{n}_{\text{no of items}}, \underbrace{W}_{\text{capacity}})$ and a subproblem knapsack (i, j) where $i \leq n$ and $j \leq W$

Recurrence

$$F(i,j) = \begin{cases} \max(\underbrace{F(i-1,j)}_{\text{exclude } i^{th} \text{ item}}, \underbrace{v_i + F(i-1, j-w_i)}_{\text{include } i^{th} \text{ item}}) & \text{if } j - w_i \geq 0 \\ F(i-1, j) & \text{if } j - w_i < 0 \end{cases}$$

eg:

Item i	Weight w_i	Value v_i
1	2	12
2	1	10
3	3	20
4	2	15

Knapsack(4, 5) where capacity = 5

Solution

w	v	i \ capacity j	1	2	3	4	5
2	12	1	0	12	12	12	12
1	10	2	10	12	22	22	22
3	20	3	10	12	22	$\boxed{30}$	32
2	15	4	10	15	25	30	37

Complexity

- Space: $\Theta(nW)$
- Time complexity: $\Theta(nW)$

- Items in optimal solution: $\Theta(n)$

Algorithm KnapSack(n, W)
// Inputs: n – no of items, W – capacity
// Output: optimal subset

// Global table F[n+1][W+1] initialised to -1
// F[0,0] initialised to 0

// Wt[n] and Val[n] global variables

```
for i = 0 to n
  for j = 0 to W
    if i == 0 or j == 0
        F[i,j] = 0
    else if j - Wt[i] >= 0:
        F[i,j] = max { F[i-1, j], Val[i] + F[i-1, j - Wt[i]] }
    else
        F[i,j] = F[i-1, j]
return F[n,W]
```

Q: Is a sequence of values in a row of the DP table for the knapsack problem is always nondecreasing?

Yes, as the capacity increases the value cannot decrease

Q: Is a sequence of values in a column of the DP table for the knapsack problem is always nondecreasing?

Yes, as the number of items increases the value cannot decrease; the previous value can be used

MEMORY FUNCTION KNAPSACK

- Bottom up advantage: each value computed only once
- Not all table entries are useful; wasted computations
- Top down disadvantage: multiple computations
- Solution: combine advantages of top down and bottom up approach

```
Algorithm MFKnapsack(i, j)
  // Inputs: i - no of items, j - capacity
  // Output: optimal subset

  // Global table F[n][W] initialised to -1
  // F[0,0] initialised to 0

  // W[n] and V[n] global variables

  if F[i,j] < 0          // not stored in table
     if j < W[i]         // item weight exceeds capacity
        value = MFKnapsack(i-1, j)

     else
        value = max (MFKnapsack(i-1, j), V[i] + MFKnapsack(i-1, j - W[i]))
        F[i,j] = value

  return F[i,j]
```

eg:

Item i	Weight w_i	Value v_i
1	2	12
2	1	10
3	3	20
4	2	15

Knapsack(4, 5) where capacity = 5

capacity j

i	0	1	2	3	4	5
0	0					
1						
2						
3						
4						

1. $F[4,5] = -1$

$j = 5 \qquad j - w_i = 5 - 2 = 3$

$\max(F[3,5],\ 15 + F[3,3])$

⋮

capacity j

i	0	1	2	3	4	5
0	0					
1						
2		-			-	
3		-	-		-	
4		-	-	-	-	

9 values not computed and filled

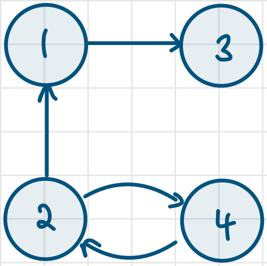
Complexity

- Space: $\Theta(nW)$
- Time complexity: $\Theta(nW)$
- Items in optimal solution: $\Theta(n)$

4. WARSHALL'S ALGORITHM

- Transitive closure of a relation
- Relations can be represented as unweighted directed graphs (edge from A to B represents that A is related to B)
- Transitivity: aRb and $bRc \Rightarrow aRc$
- Apply transitivity as many times as possible: obtain transitive closure
- Existence of all nontrivial paths in a digraph; all paths to be represented by direct edge in transitive closure

eg: Transitive closure



$$\begin{array}{c|c} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

- From source 1:

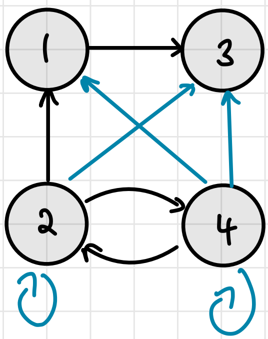
  path from 1 to 1 — NO
  path from 1 to 2 — NO
  path from 1 to 3 — YES $1 \to 3$
  path from 1 to 4 — NO

- From source 2:

  path from 2 to 1 — YES $2 \to 1$
  path from 2 to 2 — YES $2 \to 4 \to 2$
  path from 2 to 3 — YES $2 \to 1 \to 3$
  path from 2 to 4 — YES $2 \to 4$

- And so on

- Transitive closure:



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

Recurrence

- $R^{(0)} = A$ (adjacency matrix)

- $R^{(n)} = T$ (transitive closure)

- On the k^{th} iteration, the algorithm computes $R^{(k)}$

$$R^{(k)}[i,j] = \begin{cases} 1 & \text{if path from } i \text{ to } k \text{ and } k \text{ to } j \\ & \text{or } R^{(k-1)}[i,k] = R^{(k-1)}[k,j] = 1 \\ R^{(k-1)}[i,j] & \text{otherwise} \end{cases}$$

- Logical expression

$$R^{(k)}[i,j] = R^{(k-1)}[i,j] \text{ or } R^{(k-1)}[i,k] \text{ and } R^{(k-1)}[k,j]$$

Algorithm Warshall(A[n,n])

// Input: Adjacency matrix $A_{n \times n}$
// Output: Transitive closure $T_{n \times n}$

$R^{(0)} = A$

for k=1 to n

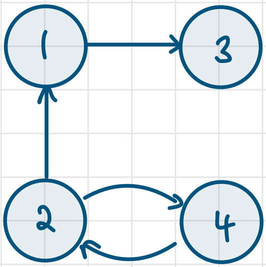
for i=1 to n
for j=1 to n

$R^{(k)}[i,j] = R^{(k-1)}[i,j]$ or $R^{(k-1)}[i,k]$ and $R^{(k-1)}[k,j]$

return $R^{(n)}$

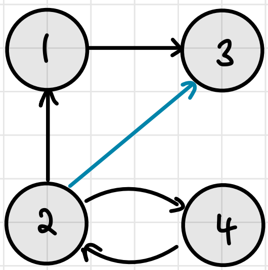
Example:

0) $R^{(0)}$



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 \end{array}$$

1) $R^{(1)}$

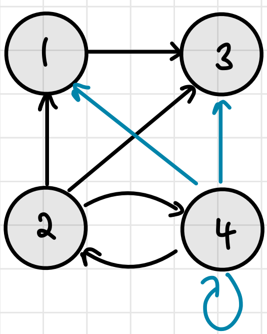


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 \end{array}$$

outgoing

incoming

2) $R^{(2)}$

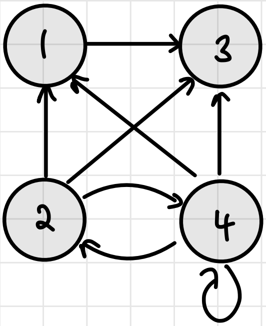


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

outgoing

incoming

2) $R^{(3)}$

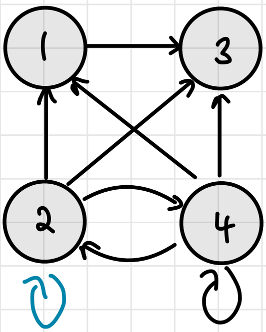


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

→ outgoing

↑ incoming

2) $R^{(4)}$



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

→ outgoing

↑ incoming

Complexity

- Time: $\Theta(n^3)$
- Space: $\Theta(n^2)$ — only 2 matrices required

Q: Is Warshall's algorithm efficient for sparse graphs?

- If adj list used?

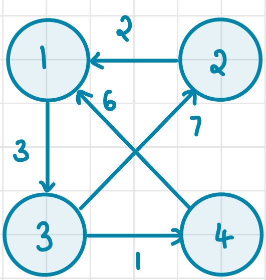
Q: Can Warshall's algorithm be used to determine if a graph is a DAG (directed acyclic graph)?

- Yes; path from node to itself - cyclic

5. FLOYD'S ALGORITHM

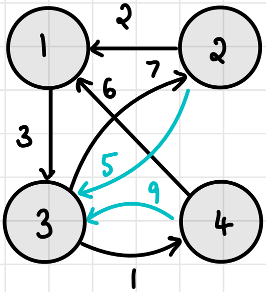
- Shortest path between every pair of vertices
- Dijkstra's: path from vertex to n-1 remaining vertices — $\Theta(n)$ paths
- Current problem: $\Theta(n^2)$ path
- Compute all pairs of shortest paths via sequence of nxn matrices $D^{(0)}, \ldots, D^{(k)}, \ldots D^{(n)}$ where $D^{(k)}[i,j]$ is the shortest path from i to j with only first k vertices allowed as intermediate vertices

eg:



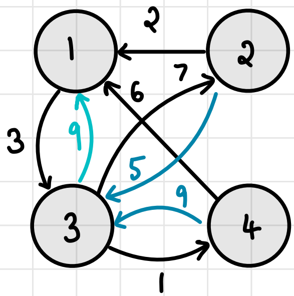
$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

via 1



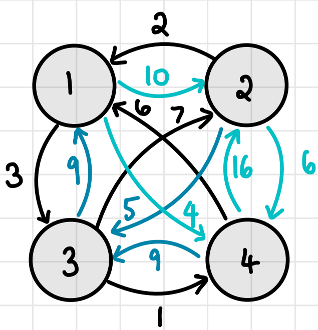
$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

via 2



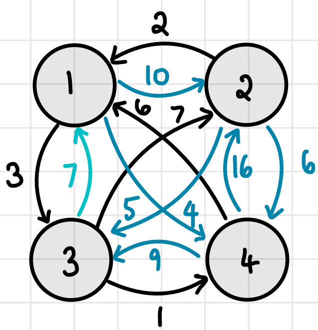
$$D^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

via 3



$$D^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

via 4



$$D^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array} \longrightarrow \text{final matrix}$$

Algorithm Floyd(A[n][n])

```
// Input: weight matrix A of a graph
// Output: Distance matrix of shortest paths

D = A

for k = 1 to n

    for i = 1 to n
        for j = 1 to n

            D[i,j] = min(D[i,j], D[i,k] + D[k,j])

return D
```

Complexity

- Time: $\Theta(n^3)$
- Space: $\Theta(n^2)$

Q: Enhance Floyd's algorithm so that shortest paths themselves and not just their lengths are found

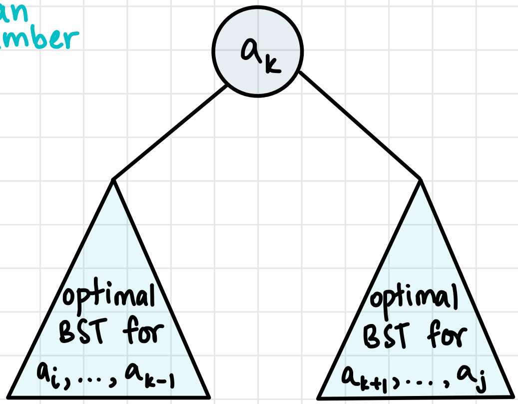
Have a second matrix PREV that stores the previous vertice visited in the path from i to j in PREV[i,j]

6. OPTIMAL BINARY SEARCH TREES

- Given n keys $a_1 < \cdots < a_n$ and probabilities $p_1, \ldots, p_n$ searching for them, find a BST with a minimum number of comparisions in successful search

- Since total number of BSTs with n nodes is given by $\frac{C(2n,n)}{n+1}$, brute force is pointless (exponential)

  Catalan number



- $C[i,j]$ — minimum average number of comparisons made in $T[i,j]$ $\longrightarrow$ tree with nodes a_i to a_j — T_i^j

- $T[i,j]$ — optimal BST for keys $a_i < \cdots < a_j$ where $1 \leq i \leq j \leq n$

$$C[i,j] = \min_{i \leq k \leq j} \Big(p_k \cdot \underbrace{1}_{\text{1 access}} + \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \text{ in } T_i^{k-1} \overbrace{+1}^{\text{root node}}) + \sum_{s=k+1}^{j} p_s \cdot (\text{level of } a_s \text{ in } T_{k+1}^{j} \overbrace{+1}^{\text{root node}}) \Big)$$

$$C[i,j] = \min_{i \le k \le j} \left\{ \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \text{ in } T_i^{k-1}) + \sum_{s=k+1}^{j} p_s \cdot (\text{level of } a_s \text{ in } T_{k+1}^{j}) + \sum_{s=i}^{j} p_s \right\}$$

Recurrence

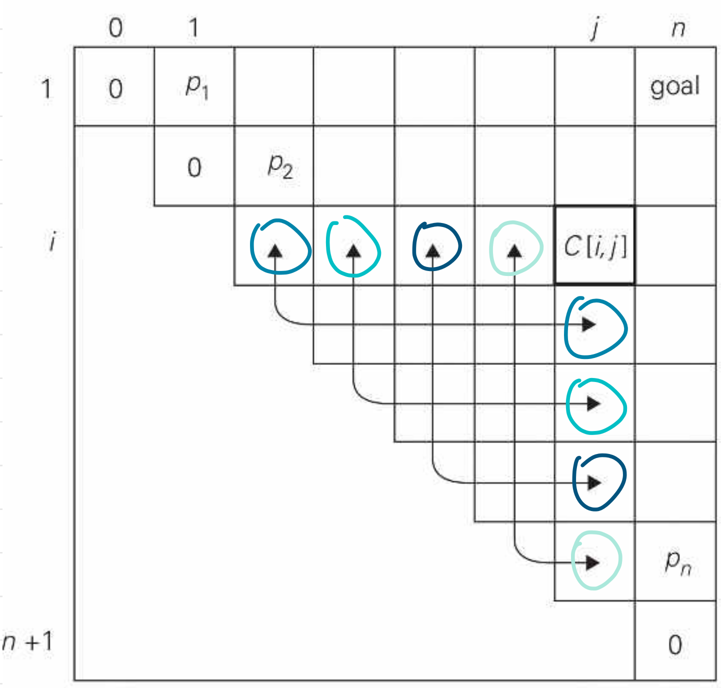
$$C[i,j] = \min_{i \le k \le j} \{ C[i,k-1] + C[k+1,j] \} + \sum_{s=i}^{j} p_s \qquad 1 \le i \le j \le n$$

one node tree

$$C[i,i] = p_i \qquad 1 \le i \le n$$

$$C[i,i-1] = 0$$

Table for DP



Eg:

	1	2	3	4	
key	A	B	C	D	$n=4$
probability	0.1	0.2	0.4	0.3	

initial tables T_0^4

main table

i \ j	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

root table

i \ j	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

Compute $C[1,2] = \min \begin{cases} k=1: & C[1,0] + C[2,2] + \sum_{s=1}^{2} p_s \\ k=2: & C[1,1] + C[3,2] + \sum_{s=2}^{2} p_s \end{cases}$

$i=1$

$j=2$

$= \min \begin{cases} k=1: & 0 + 0.2 + 0.3 = 0.5 \\ \underline{k=2}: & 0.1 + 0 + 0.3 = 0.4 \end{cases}$

root

main table

i \ j	0	1	2	3	4
1	0	0.1	0.4		
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

root table

i \ j	0	1	2	3	4
1		1	2		
2			2		
3				3	
4					4
5					

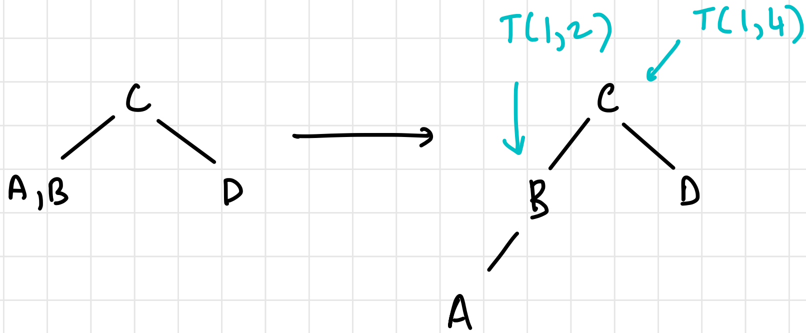
And so on,

avg no of comparisons

	j	0	1	2	3	4
i	1	0	0.1	0.4	1.1	1.7
	2		0	0.2	0.8	1.4
	3			0	0.4	1.0
	4				0	0.3
	5					0

	j	0	1	2	3	4
i	1		1	2	3	3
	2			2	3	3
	3				3	3
	4					4
	5					

Reconstruction



root obtained from root table, recursively

Algorithm

ALGORITHM *OptimalBST*$(P[1..n])$

```
//Finds an optimal binary search tree by dynamic programming
//Input: An array P[1..n] of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
//        optimal BST and table R of subtrees' roots in the optimal BST
for i ← 1 to n do
    C[i, i − 1] ← 0
    C[i, i] ← P[i]
    R[i, i] ← i
C[n + 1, n] ← 0
for d ← 1 to n − 1 do    //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j];    kmin ← k
        R[i, j] ← kmin
        sum ← P[i];    for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```

initialise comparisons and root tables

loop to find minval

avg comps.

root table

Complexity

- Time: $\Theta(n^3)$ —— can reduce to $\Theta(n^2)$
- Space: $\Theta(n^2)$

Limitations of Algorithmic Power

- There are no algorithms to solve some problems (eg: halting problem, acceptance problem)
- Certain problems can be solved in principle, but in non-polynomial time (eg: travelling salesman problem)

LOWER-BOUND ARGUMENTS

- Lower bound: an estimate on a minimum amount of work needed to solve a given problem
- Can be an exact count or an efficiency class (Ω)
- Tight lower bound: there exists an algorithm with the same efficiency as the lower bound
- Should not be possible to solve at lower complexity than lower bound — should be firm

Problem	Lower Bound	Tightness (algo exists)	
sorting	$\Omega(n \log n)$	yes	merge
search sorted array	$\Omega(\log n)$	yes	binary
element uniqueness	$\Omega(n \log n)$	yes	sort & adj (n)
integer multiplication (n bits x n)	$\Omega(n)$	unknown	
matrix multiplication (nxn)	$\Omega(n^2)$	unknown	

Strassen's $n^{2.\text{something}}$

1. Trivial Lower Bounds

- Counting no. of items to be processed in input and generated as output

Examples

(a) Max element $\longrightarrow \Omega(n)$
(b) Polynomial evaluation $\longrightarrow \Omega(n)$ for n terms
(c) Matrix multiplication $\longrightarrow \Omega(n^2)$ for each element in $n \times n$
(d) Sorting $\longrightarrow$ not best

- Note: may not always be useful

2. Adversary Arguments

- Worst case amount of work
- Imagine adversary working hard to make problem difficult to solve by adjusting input

Examples

(a) Search for element in binary search; adversary puts number in the larger of two subsets (worst case $\log n$ comparisons)

(b) Merging of two sorted list; adversary $a_i < b_j$ iff $i < j$ for $a_1, a_2, \ldots a_n$ and $b_1, b_2, \ldots, b_n$ (worst case $2n-1$ comparisons)

3. Problem Reduction

- If problem P at least as hard as problem Q then lower bound for Q is lower bound for P

- Find problem Q with known lower bound, reduce problem Q to problem P

Example

(a) P: MST for n points in cartesian plane, Q: element uniqueness problem ($\Omega(n \log n)$)

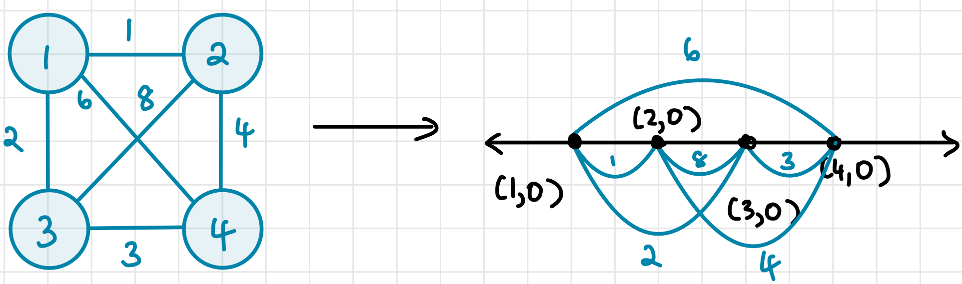
Reduce element uniqueness problem to minimum spanning tree problem (Euclidean MST problem)

Let n numbers be the n points in Cartesian plane for which MST must be found

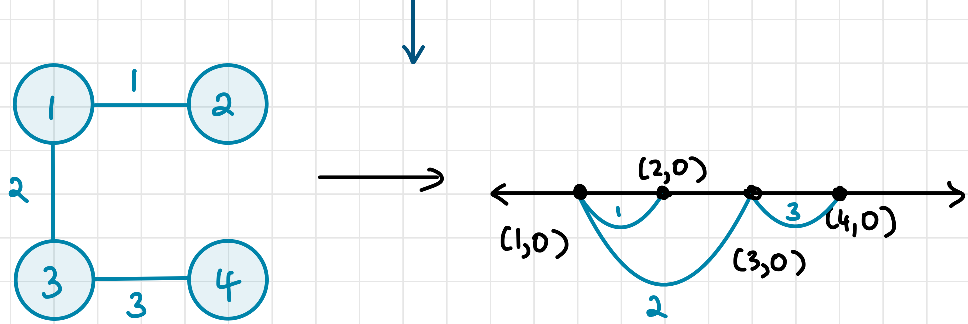
Convert n no.s to set of coordinates with $y=0$

$\{x_1, x_2, \ldots, x_n\} \longrightarrow \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$

Let T be MST of n points



MST



If 0 length edge exists, no uniqueness. Here: unique

- Prove that the classic recursive algorithm for the Tower of Hanoi puzzle makes the minimum number of disk moves

https://math.stackexchange.com/questions/2650/how-to-prove-the-optimal-towers-of-hanoi-strategy

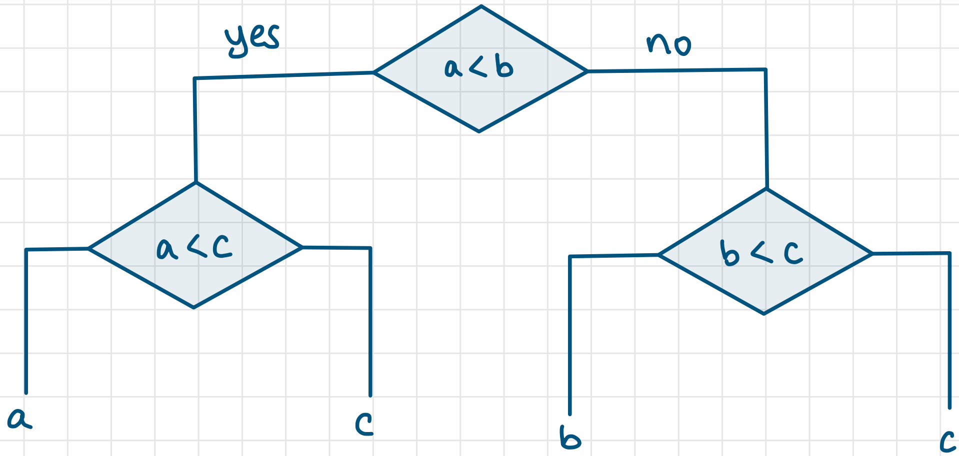
http://towersofhanoi.info/Tech.aspx

- Find a trivial lower-bound class and indicate if the bound is tight:
  - finding the largest element in an array
  - generating all the subsets of an n-element set
  - determining whether *n* given real numbers are all distinct

DECISION TREES

- Problem types: optimisation and desicion (true/false)
- Many problems can be framed in either way
- Desicion problems more convenient to study complexity
- At each node, algorithm takes decision

Eg: Decision tree for minimum of 3 no.s



- Cannot have less no. of leaf nodes than no. of solutions

Central Idea

- Tree must be tall enough for no. of leaves = no. of outcomes

- Largest no. of leaves: all leaves in last level $= 2^h$

$$\ell \leq 2^h$$

- Height must be at least $\log_2(\text{leaves})$

$$h \geq \lceil \log_2 \ell \rceil$$

1. Desicion Trees for Sorting Algorithms

- Sorting algorithms comparison-based (compare pairs of elements in list)

- Binary decision tree for comparison-based sorting to derive lower bounds on time efficiency

- Decision tree for sorting array of size n will have n! leaf nodes

$$C_{worst}(n) \geq \lceil \log_2 n! \rceil$$

Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

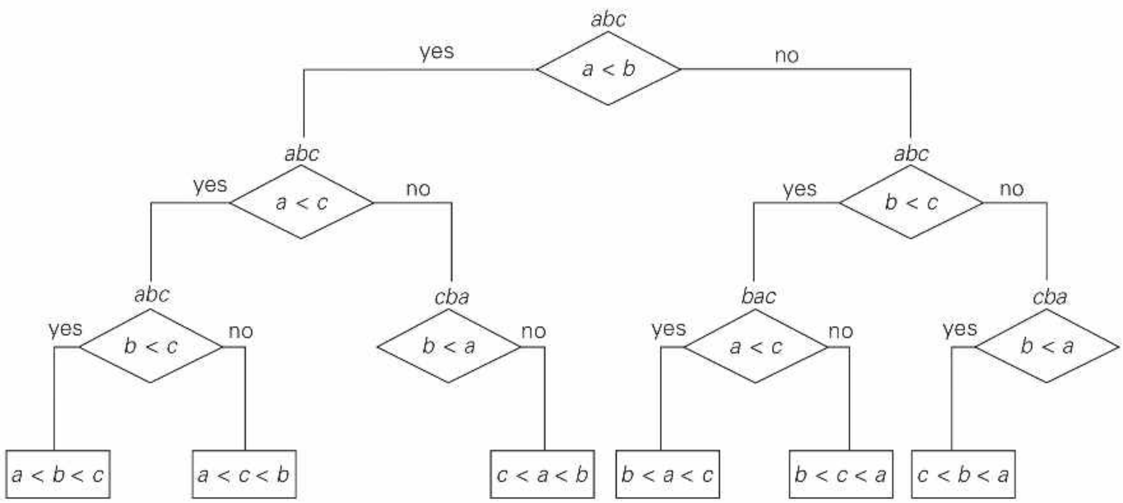
$$\left\lceil \log_2\left[(\sqrt{2\pi n})\left(\frac{n}{e}\right)^n\right] \right\rceil = \left\lceil \frac{1}{2}\log_2(2\pi n) + n\log_2\left(\frac{n}{e}\right) \right\rceil$$

$$= \left\lceil \frac{1}{2}\log_2(2) + \frac{1}{2}\log_2(\pi) + \frac{1}{2}\log_2 n + n\log_2 n - n\log_2 e \right\rceil$$

$$= \left\lceil \frac{1+\log_2 \pi}{2} + \frac{\log_2 n}{2} - (\log_2 e)n + n\log_2 n \right\rceil$$

$$= \Theta(n \log n)$$

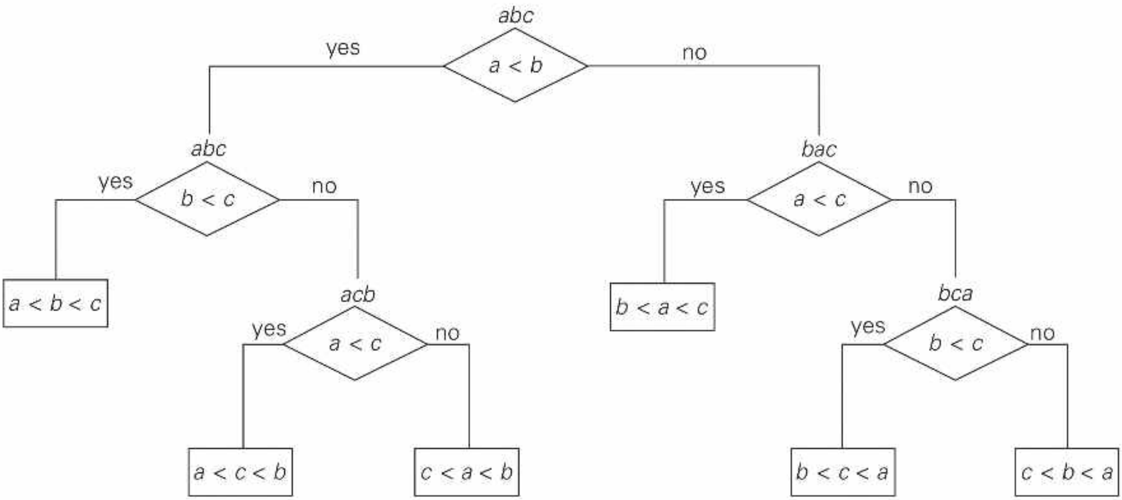
Decision Tree for 3-element Selection Sort



Average-Case Behaviour

- Average depth of leaves ; average path length from root to leaves

Decision Tree for 3-element Insertion Sort



average case:

$$\frac{2+3+3+2+3+3}{6} = 2\tfrac{2}{3} \text{ comparisons}$$

Lower Bound on C_{avg}

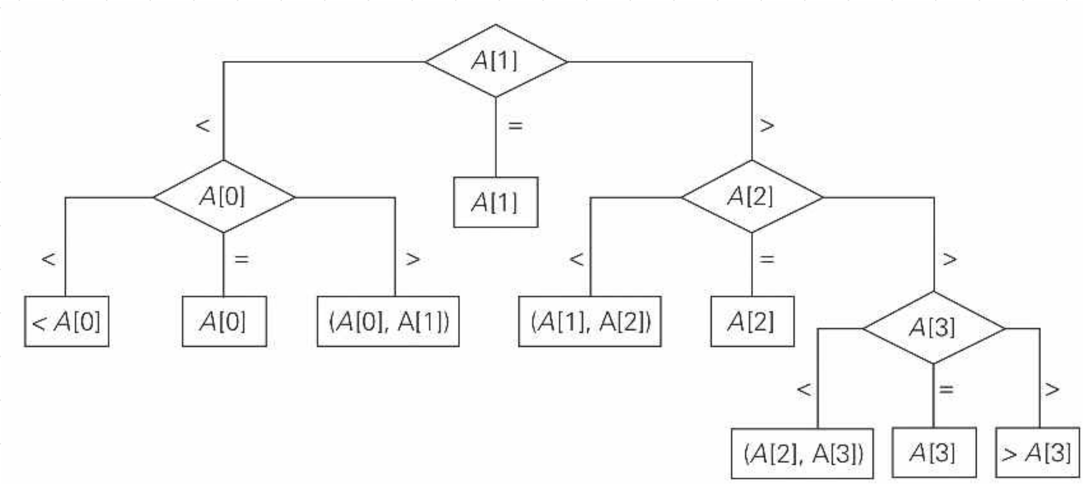
$$C_{avg}(n) \geq \log_2(n!)$$

2. Desicion Trees for Searching Algorithms

- key comparisons of array of n keys

$$C^{bs}_{worst}(n) = \lfloor \log_2 n \rfloor + 1 = \lceil \log_2(n+1) \rceil$$

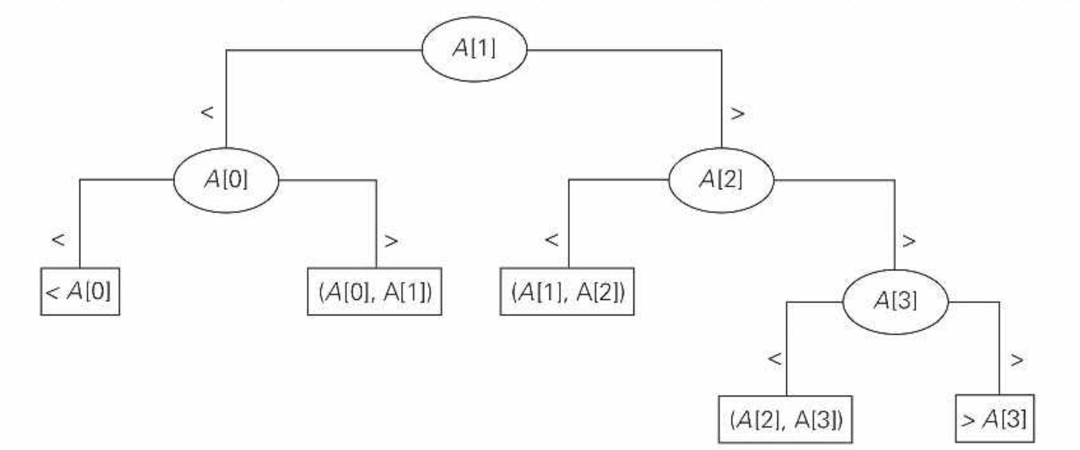
Four element Tree



$$C_{worst}(n) \geq \lceil \log_3(2n+1) \rceil$$

- Lower than $\lceil \log_2(n+1) \rceil$; tight?

Binary Decision Tree



$$C^{bs}_{worst}(n) = \lceil \log_2(n+1) \rceil$$

- Consider the problem of finding the median of a three-element set a, b, c of orderable items
  - What is the information-theoretic lower bound for comparison-based al- gorithms solving this problem?
  - Draw a decision tree for an algorithm solving this problem
  - Is the above bound tight?

COMPLEXITY CLASSES

- Is a problem tractable ; solvable in polynomial time $O(p(n))$
- Decision problems , not optimisation (for now)

class P

- Decision problems solvable in polynomial time $O(p(n))$
- Problems:
  - searching
  - element uniqueness
  - graph connectivity
  - graph acyclicity
  - primality testing - IITK https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf
- $\Theta(\log n) \in O(n)$ and $\Theta(n \log n) \in O(n^2)$ — polynomial time in big-O notation

class NP

- Nondeterministic Polynomial — Nondeterministic Turing Machine can solve in polynomial time
- Solutions can be verified in polynomial time once obtained
- Abstract two-step procedure
  - generates random string to verify
  - check if solution correct in polynomial time

BOOLEAN / CNF SATISFIABILITY

- Is a boolean function in conjunctive normal form (CNF) satisfiable (values that make the expression evaluate to 1)
- CNF: AND of ORs, i.e., POS form

Eg: $(a+\bar{b}+\bar{c})(\bar{a}+b)(\bar{a}+\bar{b}+\bar{c}) = y$

if $a=1$, $b=1$, $c=0$, check if $y=1$

Checking phase: $\Theta(n)$

Examples

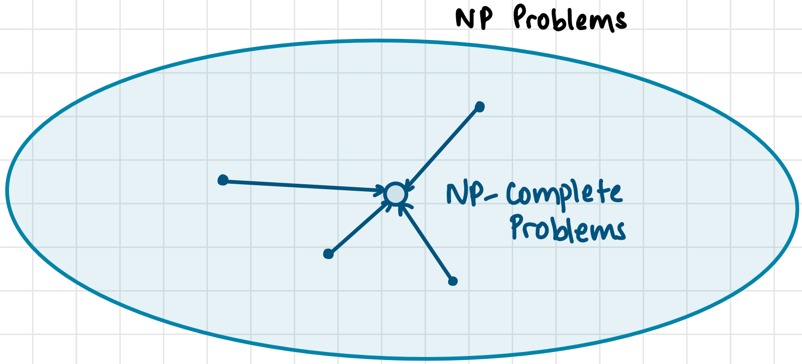
- Hamiltonian circuit existence: visit every node and come back to starting vertex
- Partition problem: possible to partition set of n integers into two disjoint subsets with same sum
- Decision variants of MST, KP, graph colouring and other combinatorial optimisation problems
- All class P problems can be solved by NP algorithm

$$P \subseteq NP$$

- Is $P = NP$ – fundamental question in CS

— class NP-complete

- A decision problem D is NP-complete if it is as hard as any problem in NP and every problem in NP is reducible to D in polynomial time

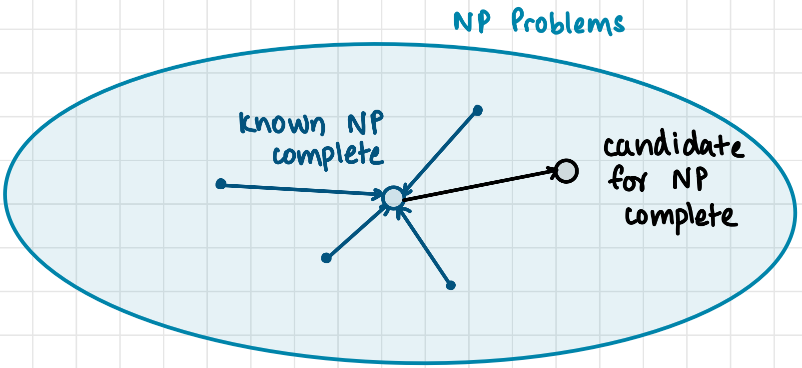


- All NP problems can be reduced to D in polynomial time

- Boolean satisfiability, Hamiltonian circuit, graph colouring, travelling salesman, subset sum are interconvertible /reducible

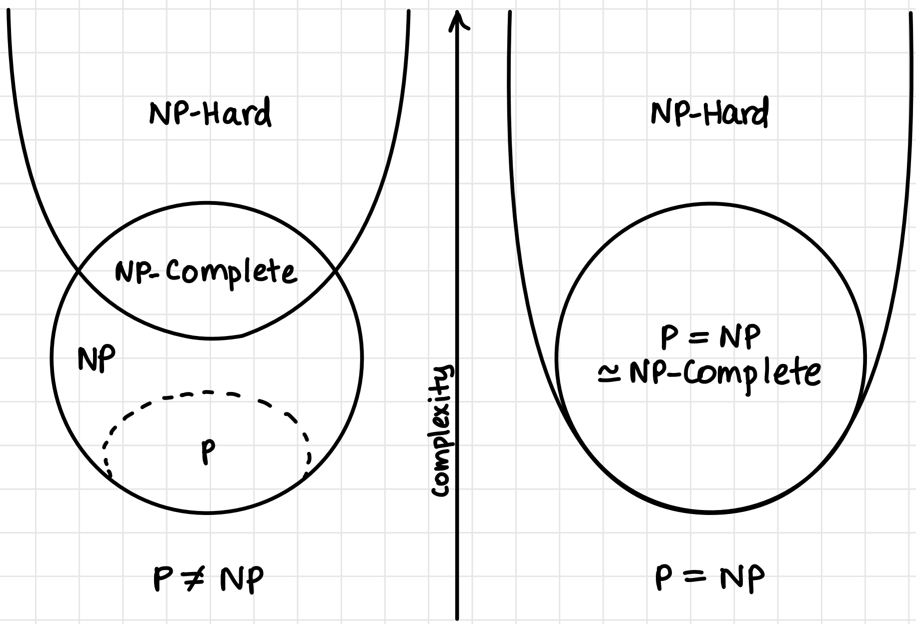
- Currently do not have polynomial time algorithm for even one of them

- Prove that no polynomial time solution exists for any one, prove for all ; prove $P \neq NP$

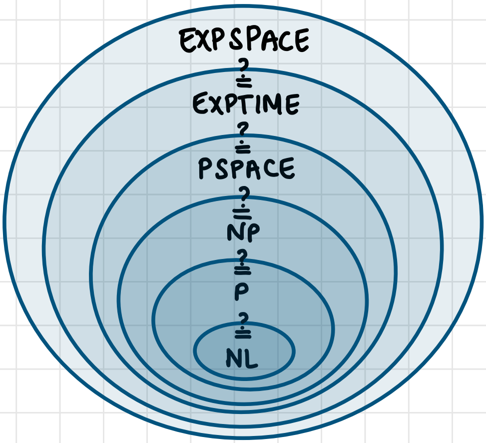


Class NP-hard

- D may or may not be in NP
- Every problem in NP polynomial time reducible to D



Complexity Hierarchy



known that at least one is a proper subset of another

$? = \longrightarrow$ unknown

BACKTRACKING

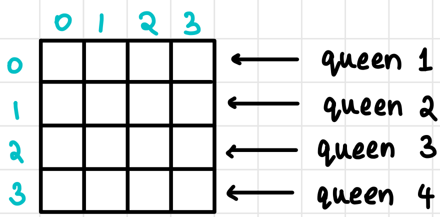
- When polynomial solutions for combinatorial problems do not exist
- Smart ways of exploring solution space (better than exhaustive solution)
- Worst case still exponential ; eliminates unnecessary cases from exhaustive search
- Further: branch and bound

Steps

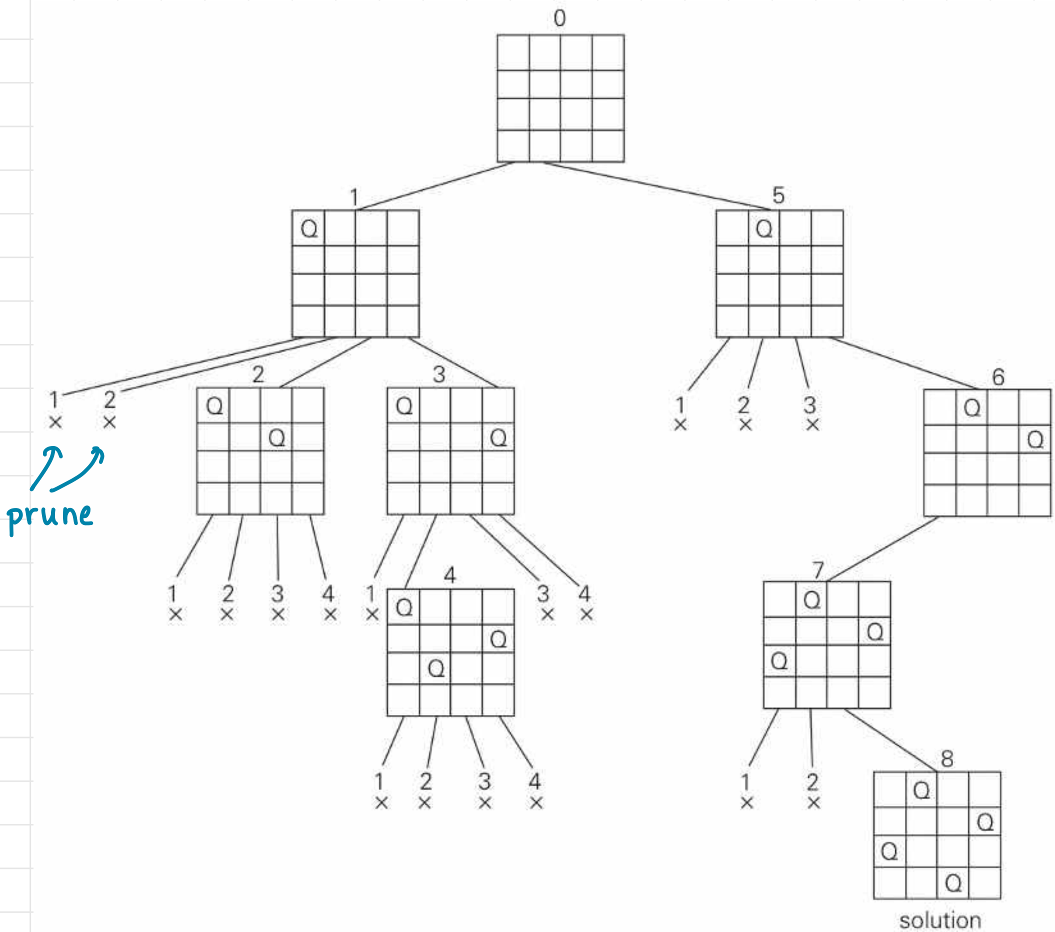
- Construct state-space tree — nodes: partial solutions and edges : choices in extending partial solutions
- Explore using DFS
- Prune nonpromising nodes (DFS stops and backtracks)

N-Queens Problem

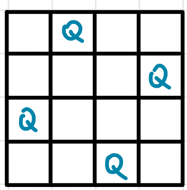
- Place N queens on an NxN chess board so that no two of them are in the same row, column or diagonal



- Find column numbers for each queen
- No solution for 2x2, 3x3

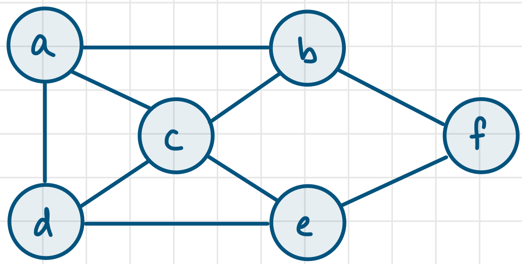


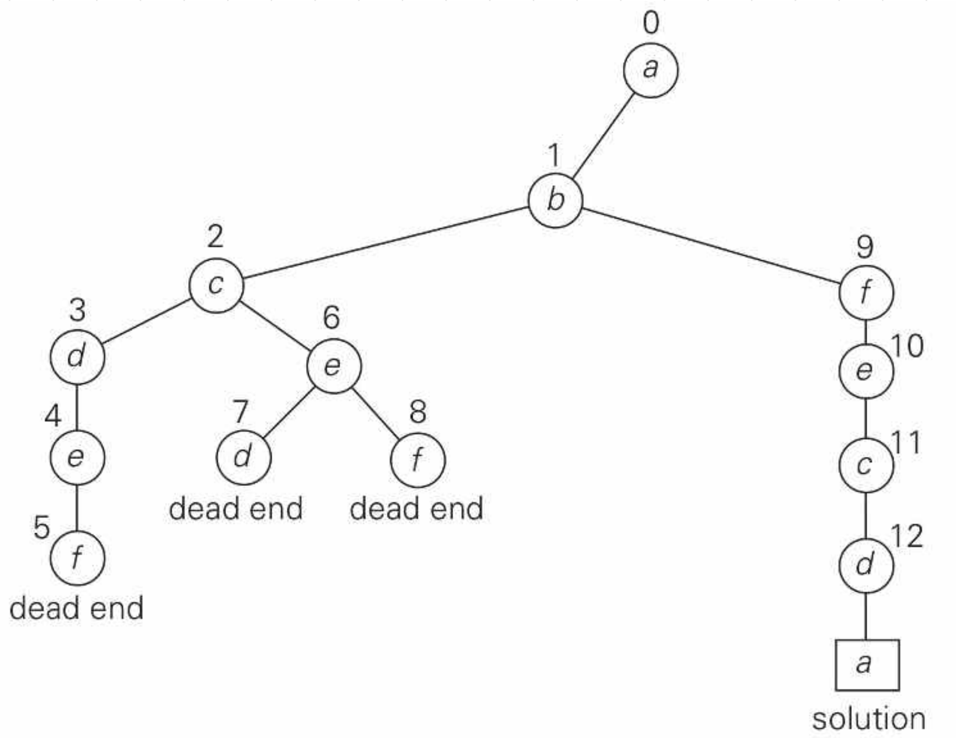
- Stop if columns equal or diagonals equal



Hamiltonian Circuit

- cycle in a graph that passes through all vertices of graph exactly once
- Source node does not matter

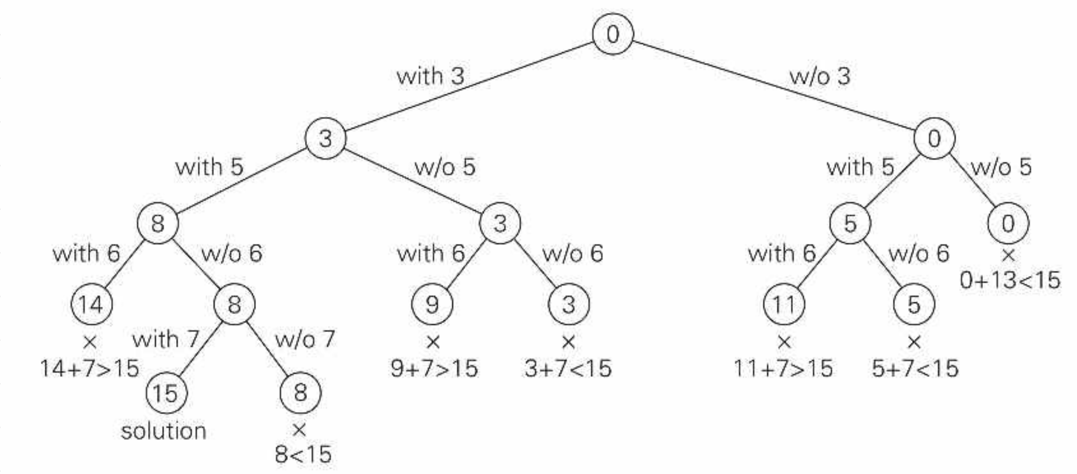




Subset Sum Problem

- Set $A = \{a_1, a_2, \ldots, a_n\}$ of n positive integers, find subset whose sum is equal to given positive integer d

Eg: $A = \{3, 5, 6, 7\}$, $d = 15$



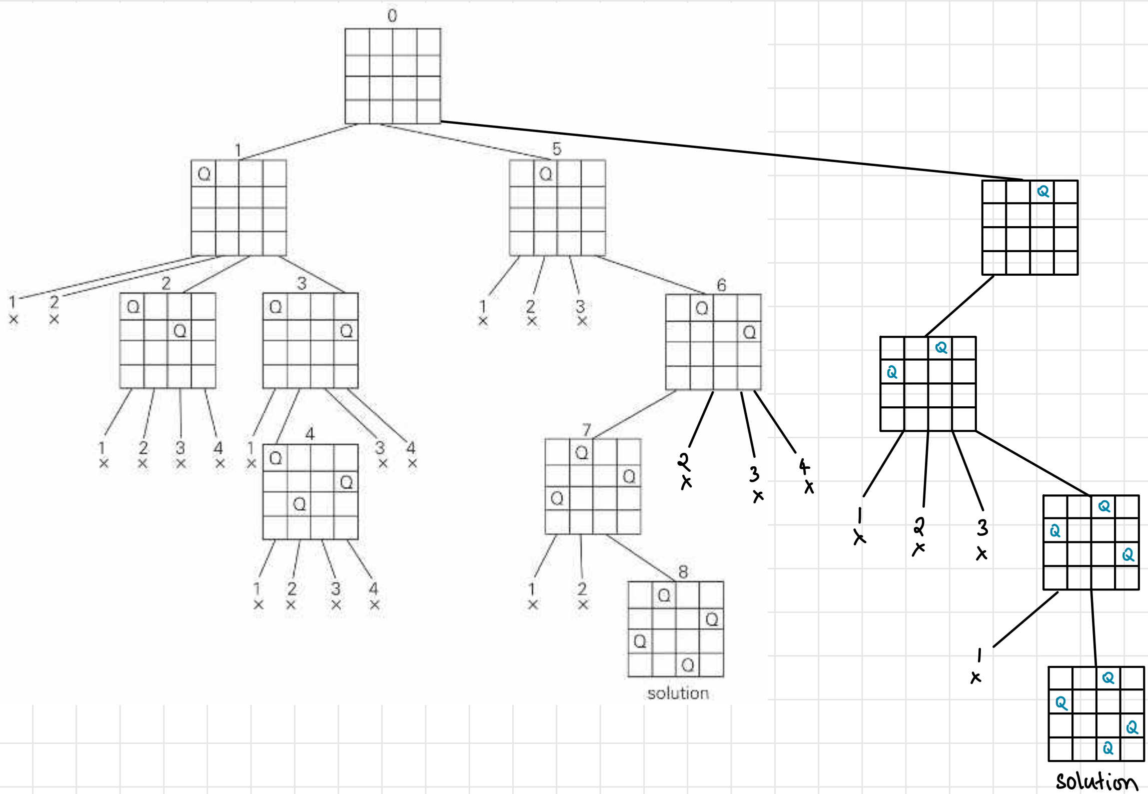
GENERAL BACKTRACKING ALGORITHM

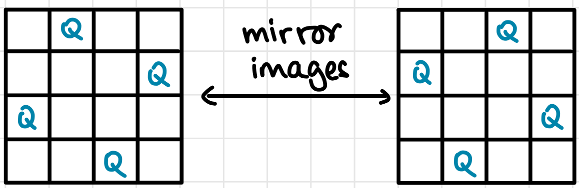
Algorithm Backtrack(X[1...i])
// Input: first i promising components of solution
// Output: all tuples in solution $(x_1, x_2, \ldots x_n)$

if X[1...i] is solution
    write X[1...i]

else
    for each element $x \in S_{i+1}$ and constraints
        X[i+1] = x
        Backtrack(X[1...i+1])

- Continue the backtracking search for a solution to the four-queens problem, to find the second solution to the problem
- Explain how the board's symmetry can be used to find the second solution to the four-queens problem





BRANCH & BOUND

- Improvement upon backtracking
- Best value of objective function on any solution that can be obtained by adding further components to the partially constructed solution at node

Termination

- Value of bound (upper/lower) of node not better than best solution seen so far
- No feasible solution as constraints already violated
- No further choices - compare

Job Assignment Problem

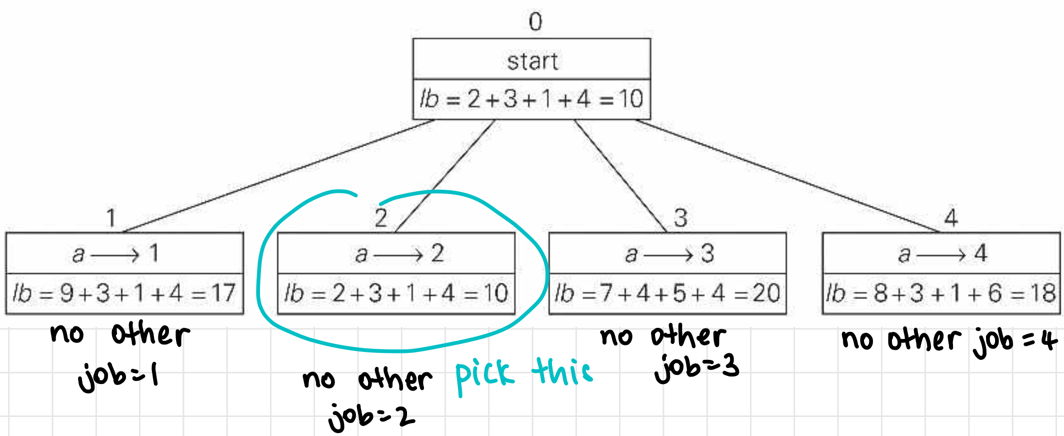
- Cost minimised ; lower bound
- Lower bound: sum of each person's lowest cost jobs (usually not a solution; acts as lower bound)

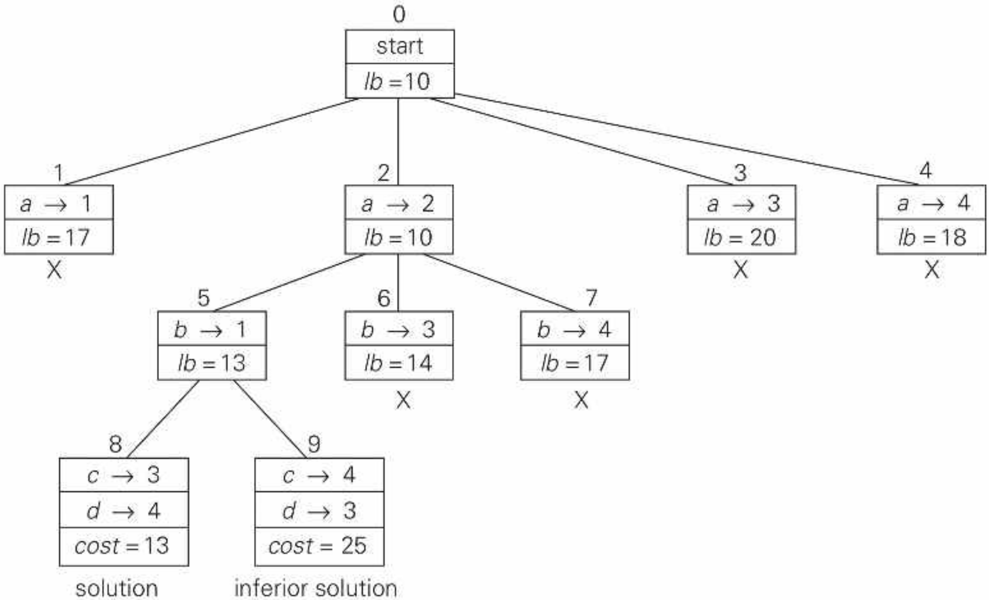
$$
\begin{array}{ccccccc}
 & \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} & \\
C = & \left[\begin{array}{cccc} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{array}\right] & & & & \begin{array}{l} \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{array}
\end{array}
$$

lower bound = 2+3+1+4 = 10

State Space Tree

- Best-first branch and bound
- Generate all children, go to best child





Knapsack Problem

Item i	Weight w_i	Value v_i	$\frac{\text{value}}{\text{weight}}$
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

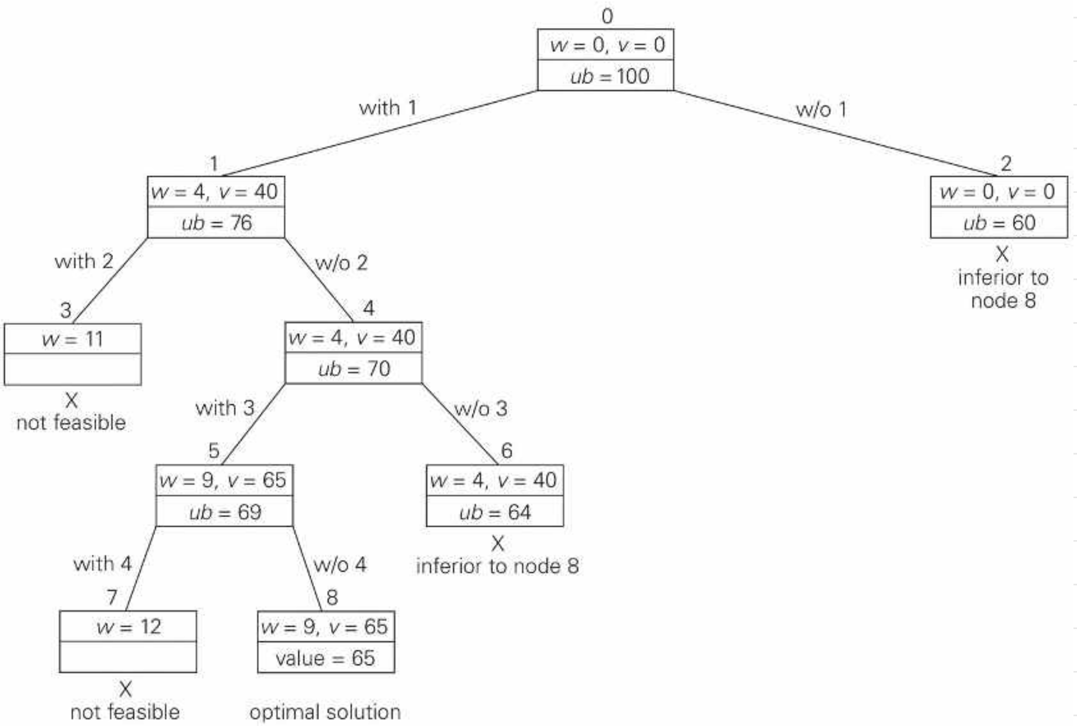
Knapsack(4, 10) where capacity = 10

- Desire: max value & min weight $\Rightarrow \frac{\text{value}}{\text{weight}}$
- Arrange in descending order
- Upper bound

$$ub = \underbrace{v}_{\substack{\text{current value of} \\ \text{items } 1 \text{ to } i}} + \underbrace{(W - w)}_{\text{space left}} \underbrace{\left(\frac{v_{i+1}}{w_{i+1}}\right)}_{v/w \text{ of next item}}$$

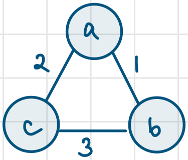
Descending Order

Item i	Weight w_i	Value v_i	$\frac{\text{value}}{\text{weight}}$
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4



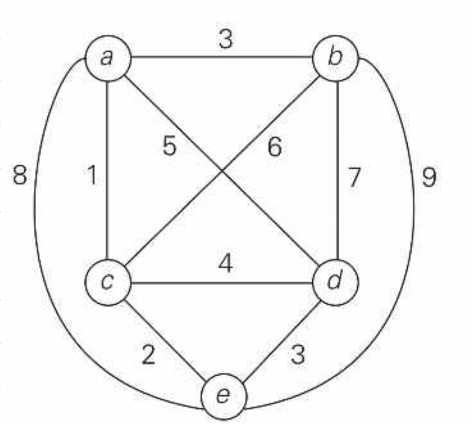
Travelling Salesman Problem

- start in a city, complete Hamiltonian circuit on weighted graph
- Lower bound: sum of costs of 2 lowest edges at a node and then divide by 2

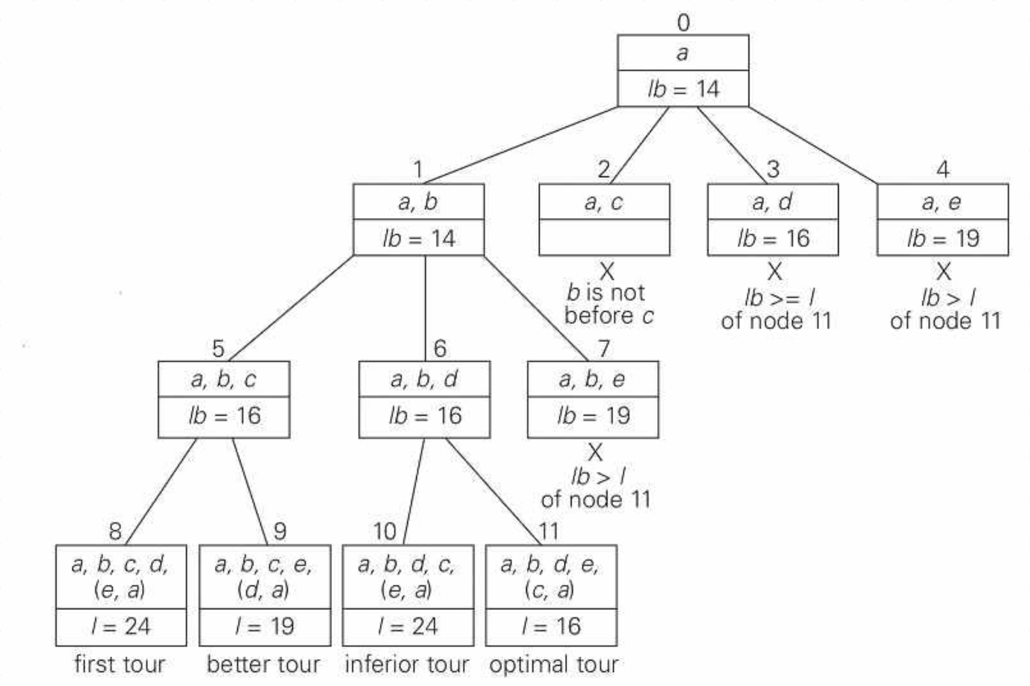


$$s = (2+1) + (1+3) + (2+3) = 12$$
$$lb = \left\lceil \frac{s}{2} \right\rceil$$

Graph



Tree



- mirror image: same cost