Brute Force Algorithms

- Try *every* possible solution
- Exhaustive search
- Ex:
  - linear search
  - every pair (closest pair, selection sort)
- May involve *generating* every possible solution
- Typically: infeasible for even "moderately" sized inputs

Generating Combinations

- Combinations are: unordered subsets

- $\binom{n}{k} = O(n^k)$

- $A = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{d}\}$ $n = 4$ $\binom{4}{2} = \frac{4!}{2!\,2!} = 6$

$K = 2$ combinations

$\{\overset{1}{a}, \overset{2}{b}\}$, $\{b, c\}$ $\{c, d\}$ $(= \{d, c\})$

$\{a, c\}$ $\{b, d\}$

$\{a, b\}$ $\quad$ 2 4

- Observation: it suffices to generate combinations of elements from $\{1, 2, 3, \ldots, n\}$

- numbers can represent indexes

- Given n, k: generate all k-combinations of $\{1, 2, \ldots n\}$
  - Start with $\{1, 2, \ldots, k\}$
  - current combination: $a_1 a_2 \ldots a_k$
  - Want to generate the <u>next</u> k-combination.
  - Locate the last element a_i such that

$$a_i \neq n - k + i$$

- Replace a_i with $a_i + 1$
- replace a_j with $a_i + j - i$ for $j = i+1, i+2 \ldots k$

$n = 5 \quad \{1, 2, 3, 4, 5\}$

$k = 3$

$\quad a_1 \; a_2 \; a_3$

current: $1, 4, 5 \longrightarrow 2, 3, 4$

next: $\boxed{a_1} = 1 \overset{?}{=} n - k + i$

$5 - 3 + 1$

$\neq 3$

$a_2 = 4 \overset{?}{=} n - k + i$

$= 5 - 3 + 2$

$= 4$

$a_3 = 5 \overset{?}{=} n - k + i$

$= 5 - 3 + 3$

$= 5$

replace a_i with $a_i + 1$

$a_1 \longrightarrow \boxed{2}$

$a_2 \longrightarrow a_i + j - 1$

$= 2 + 2 - 1$

$= \boxed{3}$

$a_3 \longrightarrow a_i + j - 1$

$= 2 + 3 - 1$

$= \boxed{4}$

$$\begin{array}{l} \quad\quad\;\; a_1 \;\; a_2 \;\; a_3 \\ \text{curr} = 2, 3, 4 \longrightarrow 2, 3, 5 \end{array}$$

replace a_3 with $a_3 + 1$

$a_3 \to 5$

$a_1 = 2 \stackrel{?}{=} n - k + i$

$5 - 3 + 1$

$\neq 3$

$a_2 = 3 \stackrel{?}{=} n - k + i$

$5 - 3 + 2$

$\boxed{\neq} 4$

$a_3 = 4 \stackrel{?}{=} 5 - 3 + 3$

$\boxed{\neq} 5$

$a_1 \quad a_2 \quad a_3$

$2, 3, 5$

$a_1 = 2 \overset{?}{=} n - k + i$

$= 5 - 3 + 1$

$\neq 3$

$a_2 = 3 \overset{?}{=} 5 - 3 + 2$

$\neq 4$

$a_3 = 5 \overset{?}{=} 5 - 3 + 3$

$= 5$

$i = 2$

replace a_2 with $a_2 + 1$

$a_2 \to 4$

should be i

replace a_j with $a_i + j - \cancel{1}$ for $j = i+1 .. k$

$\cancel{a_3 \to 4 + 3 - 1 = 6}$

$a_3 \to a_i + j - i$

$= 4 + 3 - 2$

$= 5$

$\cancel{7.}$

next one: $2, 4, 5$

Generating Permutations

- Given n, generate all permutations of $1, 2, 3, \ldots n$

- $n = 3$

  123 ← identity permutation
  132
  213 $\quad 3! = 6$
  231
  312
  321

  $n = 4 \quad n! = 24$

  $$\left.\begin{array}{c} 1234 \\ 1243 \\ 1324 \\ \vdots \\ 4321 \end{array}\right\} 24$$

- $n!$ total permutations

- Johnson - Trotter algorithm

- start with $1, 2, 3, \cdots n$

- current: $a_1 a_2 \cdots a_n$

- Find the right most pair $a_i\ a_{i+1}$ such that $a_i < a_{i+1}$

- Find the smallest element to the right larger than a_i (call it a')

swap a'

$$a_1\ a_2\ \cdots\ a_i < \underbrace{\overbrace{a_{i+1}\ a_{i+2}\ \cdots\ a_n}^{\text{sort}}}_{\text{smallest element larger than } a_i}$$

- swap a_i, a'

- order (sort) elements to the right of a'

$n = 6$, 1 2 3 4 5 6

swap

curr: 1 6 3 5 4 2

→ rightmost pair $a_i\ a_{i+1}$ such that $a_i < a_{i+1}$

$a_i = 3$

$a' = 4$

swap a_i a' : swap 3, 4 ⟶ $1\ 6\ 4\ \underbrace{5\ 3\ 2}_{\text{sort}}$

↓

1 6 4 2 3 5

164235

$a_i = 3$

$a' = 5$

swap a_i a' → 164253
sort

swap
164 2 5 3

$a_i = 2$ swap a_i a' → 164352 → 164325
sort

$a' = 3$

⋮

6 5 4 3 2 1

~~15!~~ ~~20!...~~

$52! = ?$ $\qquad\qquad$ $10!$

$8.065\ldots \times 10^{67}$

5 billion people shuffled a deck once per second for 1000 years

what percentage of suffles have been done?

$$1.95 \times 10^{-48} = \underbrace{.00\,\cdots}_{4\cancel{8}\ \text{zeros}}0195\ \%$$

2

150 peta FLOPs . .

10^{15}

Step 1: choose a set of size k

$$\binom{n}{k}$$

Step 2: arrange the k elements

$$k!$$

in total: $\binom{n}{k} \cdot k!$

$$= \frac{n!}{(n-k)!\,\cancel{k!}} \cdot \cancel{k!}$$

$$P(n,k) = \frac{n!}{(n-k)!}$$

Input: n, k

Output: ~~gen~~ all k-permutations of $1, 2, \cdots n$

for each k combination A

for each permutation π of A:

output π

$$\frac{n!}{(n-k)!}$$

$$\sum_{k=0}^{n} \frac{n!}{(n-k)!}$$

Set Partitions:

Given a set $A = \{a_1 \cdots a_n\}$

how many ways are there to partition A into

disjoint subsets

$A = \{a, b, c\}$ 5

$A = \{a, b, c, d\}$

$\{a, b, c\}$ — $\{a, b, c, d\}$, $\{a, b, c\}, \{d\}$

$\{a\}\ \{b, c\}$ — $\{a, d\}\ \{b, c\}$, $\{a\}, \{b, c, d\}$, $\{a\}, \{b, c\}, \{d\}$

$\{a, c\}, \{b\}$ — 3

$\{a, b\}\ \{c\}$ — 3

$\{a\}\ \{b\}\ \{c\}$ — 4

15

The number of set partitions corresponds to the Bell Numbers:

$\{a\}$

$\{a,b\} \to \{a,b\}$

$\{a\}\{b\}$

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

1, 1, 5, 15, ...

?

$n = 6$ $\qquad 6!$ $\qquad$ 123456

123465

$\vdots$

654321

From a set of size n, arrange k of them
order

$$P(n,k) = P_k^n = {}_nP_k \qquad n=6,\ k=3$$

$$P(6,3) =$$

Generate permutations with repetition

$\{A, G, C, T\} \longrightarrow \{a_1 a_2 \cdots a_n\}$ total: n^K

generate all K permutations w/ repetitions

$$\left.\begin{array}{c} AAA \\ AAG \\ AAC \\ AAT \\ AGA \\ AGG \\ AGC \\ \vdots \end{array}\right\} 64$$

TTT

$\boxed{4}\,\boxed{4}\,\boxed{4} = 64 = 4^3$

$n = 2 \longrightarrow \{0, 1\}$

bit strings of length $k \longrightarrow 2^k$

$k = 3$

$000 \longrightarrow 0$
$001 \quad 1$
$010 \quad 2$
$011 \quad 3$
$100 \quad 4$
$101 \quad 5$
$110 \quad 6$
$111 \quad 7$

to generate k permutations with repetition:

count in base n

from $0 \ldots n^k - 1$

$n = 2$ binary

$n = 4$ (DNA)

$n = 8$ octal

$n = 16$ hex

Satisfiability

- A boolean predicate on n variables

$$\vec{x} = x_1, x_2 \cdots x_n$$

$$P(\vec{x}) = P(x_1, x_2, \ldots, x_n)$$

- Ex:

$$P(x_1,x_2) = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

x_1	x_2	value
0	0	1
0	1	0
1	0	0
1	1	1

- Quantify:

$$\exists x_1, x_2 \left[P(x_1, x_2) \right] \longrightarrow$$ Does there exist a truth assignment for x_1 and x_2 such that

$$P(x_1, x_2) \equiv 1$$

"evaluates to true"

Problem:

Given a boolean predicate $P(x_1\ x_2 \cdots x_n)$ on n variables

Output: true if there exists a satisfying assignment to $x_1 \cdots x_n$ that makes P true

- $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

- generate all bit strings (permutations with repetition) and <u>test</u>

001

000 truth tables

~~001~~

010

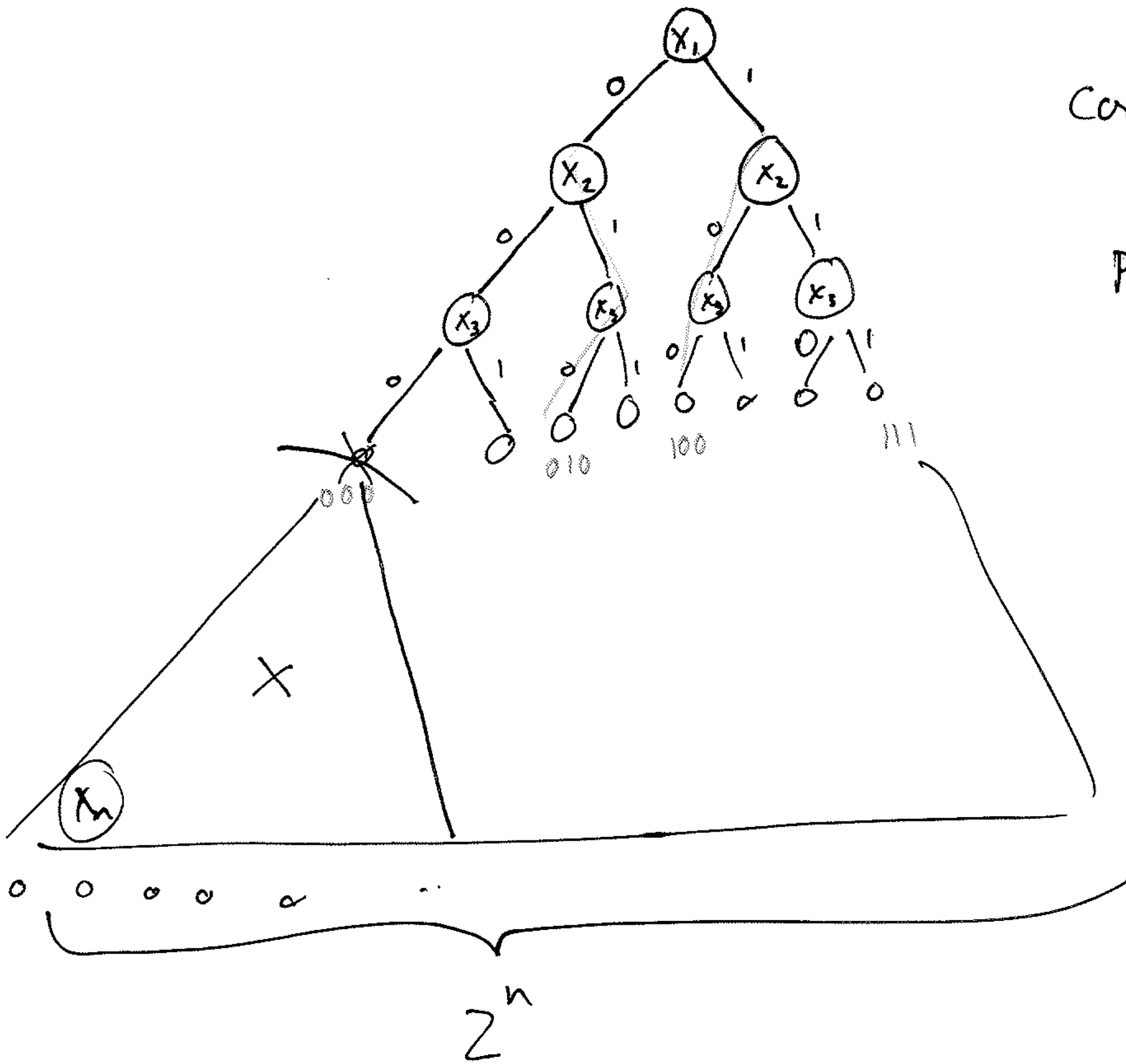
⋮

111

0

0

$\Theta(2^n)$



Computation tree

prune the tree!

Goal: design an algorithm to recursively generate all possible truth assignments to a given predicate

Start with no truth assignments

⋮

suppose at some point in the recursion we have a partial truth assignment

$$\begin{matrix} x_1 & x_2 & & x_k & x_{k+1} \cdots x_n \\ \downarrow & \downarrow & & \downarrow & \\ t_1 & t_2 & \cdots & t_k & \underbrace{\qquad\qquad}_{\text{undetermined}} \end{matrix}$$

- ~~test: if P(x.~~
- next step: set t_{k+1} to false, recurse

  if satisfiable, stop. else

  reset t_{k+1} to true and recurse

Satisfiability: Iterative Brute Force

Input: **A predicate** $P(\vec{x})$

Output: true if P is satisfiable, false otherwise

For each truth value $\vec{t} = t_1 t_2 \cdots t_n$ $\rightarrow O(2^n)$

if ($P(\vec{t}) \equiv 1$)

output true

output false

- $x_1\ x_2\ x_3 \cdots x_n$

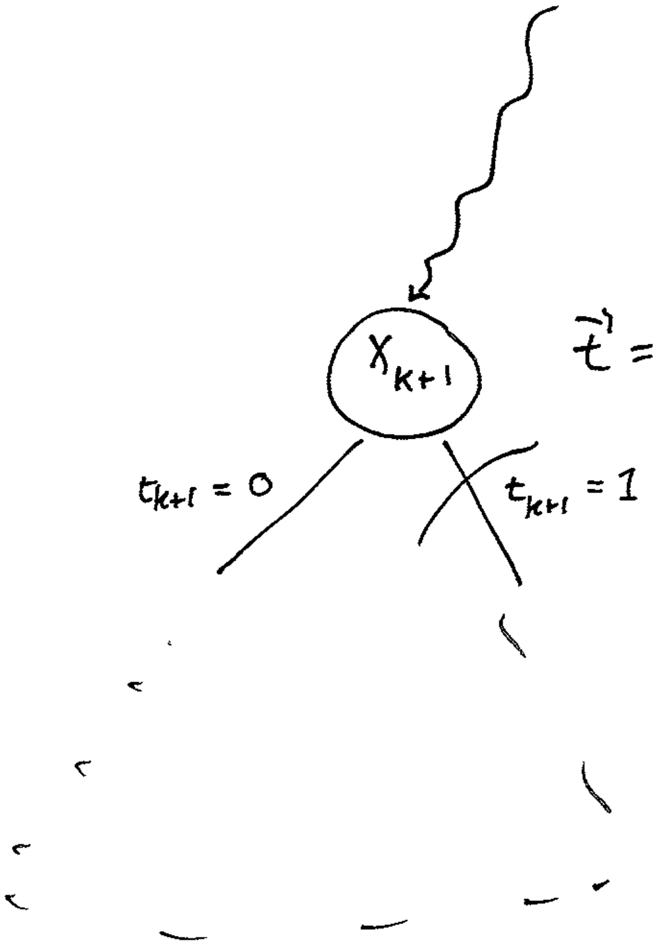
  $0\ \ 1\ \ \underbrace{t_3 \cdots t_n}_{2^{n-2}}$

- Suppose you have a subroutine

$$\text{PartialSAT}(P, \vec{t}) = \begin{cases} 0 & \text{if } P \text{ evalutes to false on } \vec{t} \\ 1 & \text{if } P \text{ evaluates to true on } \vec{t} \\ \phi & \text{otherwise} \end{cases}$$

P: predicate

$\vec{t}$: partial truth assignment



$$\vec{t} = \underbrace{t_1 t_2 \cdots t_k}$$

SAT$(P, \vec{t})$

Input: A predicate $P(x_1 \cdots x_n)$

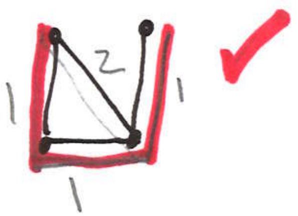
a partial truth assignment $\vec{t} = t_1 t_2 \cdots t_k$

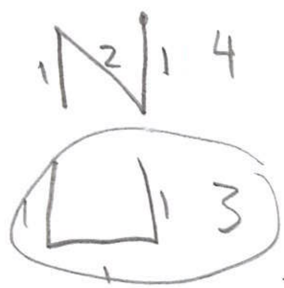
Output: true if P is satisfiable, false otherwise

```
if (k = n)
  return P(t⃗)
else
  if ( PartialSAT(P, t⃗) = 0 )
        // prune: stop the recursion
        return false
  else if ( PartialSAT(P, t⃗) = 1 )
        // prune
        return true
  else if ( PartialSAT(P, t⃗) = ∅ )
        // the predicate may still be satisfiable or not
        t_{k+1} ← 0
        if ( SAT(P, t⃗) = 1 )
          return true
        else
          t_{k+1} ← 1
          return SAT(P, t⃗)
```

Given: A graph $G = (V, E)$ (undirected)

Output: true if G contains a Hamiltonian Path, false otherwise



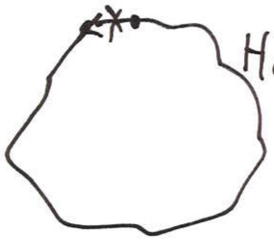


<u>Defn</u> A Ham. Path in a graph G is a simple path that traverses every <u>vertex</u> exactly once

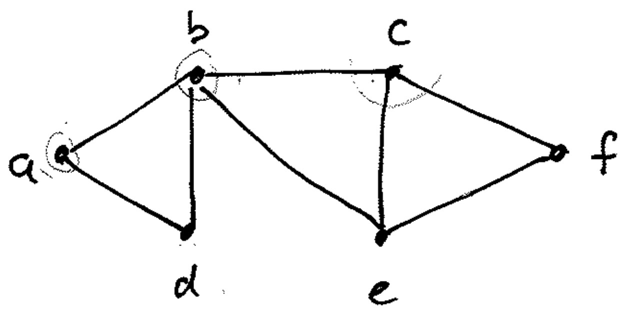
<u>no</u> Ham. path

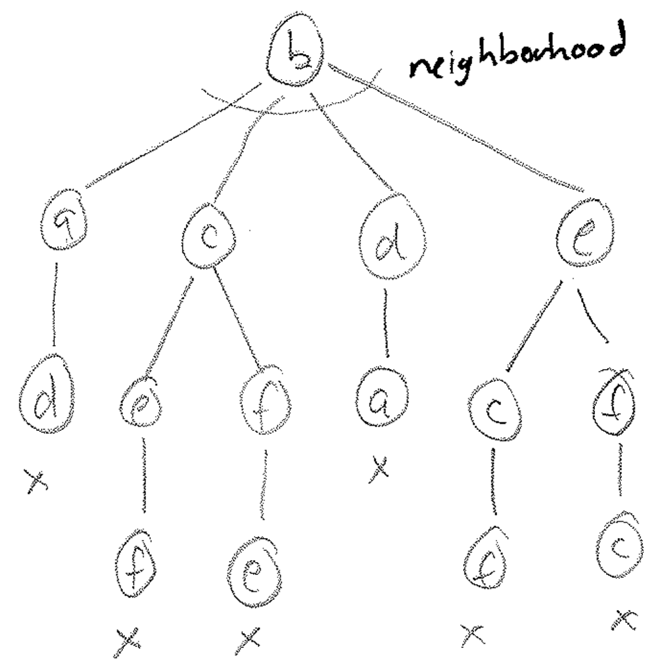
Variations:

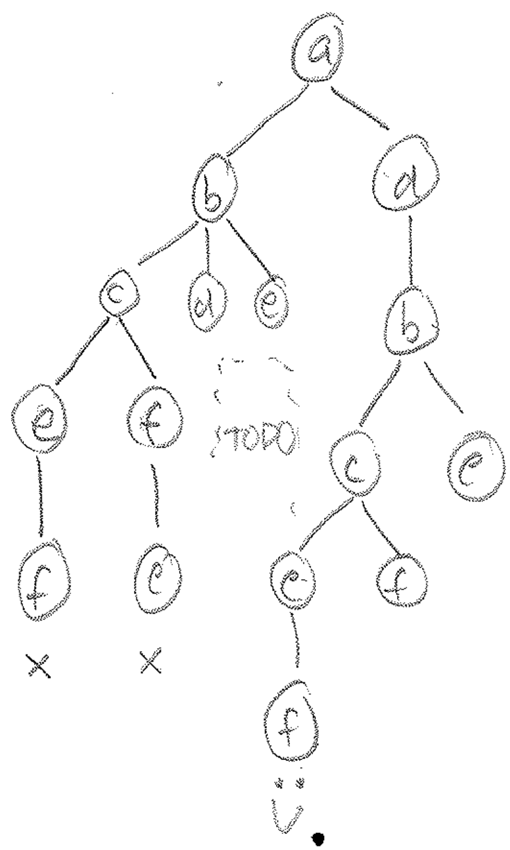
- Hamiltonian Cycle
- Weighted Ham. Path / Cycle
- TSP = Traveling Sales Person Problem.



Ham. Cycle







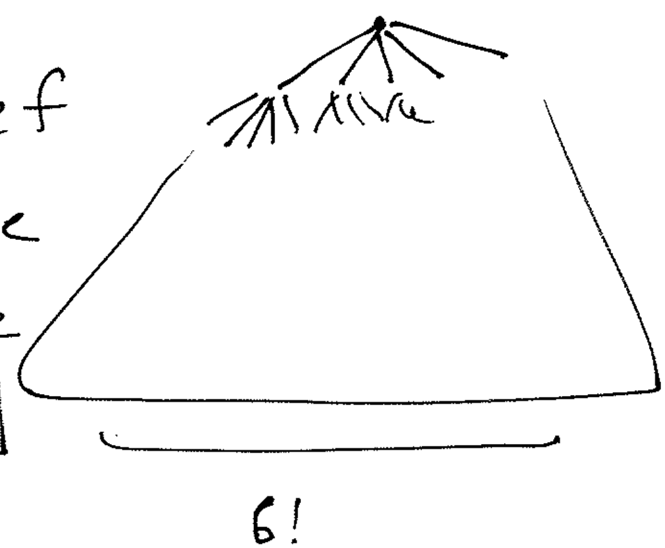
a b c d e f

a b cd f e

cd a b f e

a b f e cd

d c



Ham-Path Brute Force Iteration:

Input: A graph $G = (V, E)$

Output: true if G contains a Ham Path, false otherwise

for each permutation of vertices $\pi = v_1 v_2 \cdots v_n$ $O(n!)$

    isPath ← true
    for $i = 1 \ldots n-1$
        if ($(v_i, v_{i+1}) \notin E$)
            isPath ← false
    if(isPath)
        output ~~yes~~ true

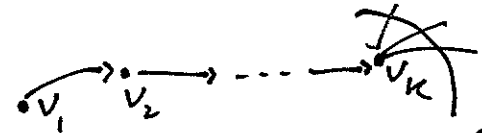
output false

abc
acb
bac
bca
cab
cba

HamWalk(G, p)

Input: A graph $G = (V, E)$, a (partial) path $p = v_1 v_2 \cdots v_K$

Output: true if G contains a Ham. Path, false otherwise



$N(v_K)$ = neighbourhood of v_K

```
if (k = n)
  output true
else
  foreach v ∈ N(v_K)
    if (v ∉ p)
      HamWalk(G, p + v)
output false
```

Ham - Main (G)

Input: A graph $G = (V, E)$

Output: true if there is a Ham. Path ~~starting~~ at any vertex

for each vertex $v \in V$

L HamWalk (G, v)

output false

0-1 Knapsack Problem

Given: A collection $A = \{a_1 \cdots a_n\}$ of objects and

a weight function $wt : A \to \mathbb{R}^+$

and value function $val : A \to \mathbb{R}^+$

and an overall capacity W

Output: A subset $S \subseteq A$

such that

$$wt(S) = \sum_{s \in S} wt(s) \leq W$$

and

$$val(S) = \sum_{s \in S} val(s) \text{ is maximized}$$

$O(2^n)$

Item	Value	Wt
a_1	15	1
a_2	10	5
a_3	9	3
a_4	5	4

$W = 8$

$S = \{a_1\ a_2\ a_3\ a_4\}$

value : 39

wt : 13 infeasible solution

$S = \{a_1, a_2\}$

value : 25

wt : 6

feasible, optimal ?

$S = \{a_1\ a_3\ a_4\}$

value : 29

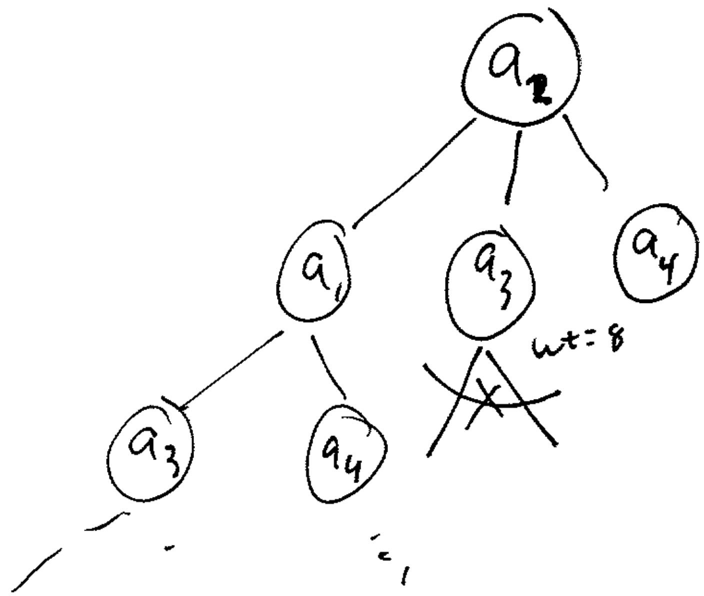
wt : 8

feasible better Than

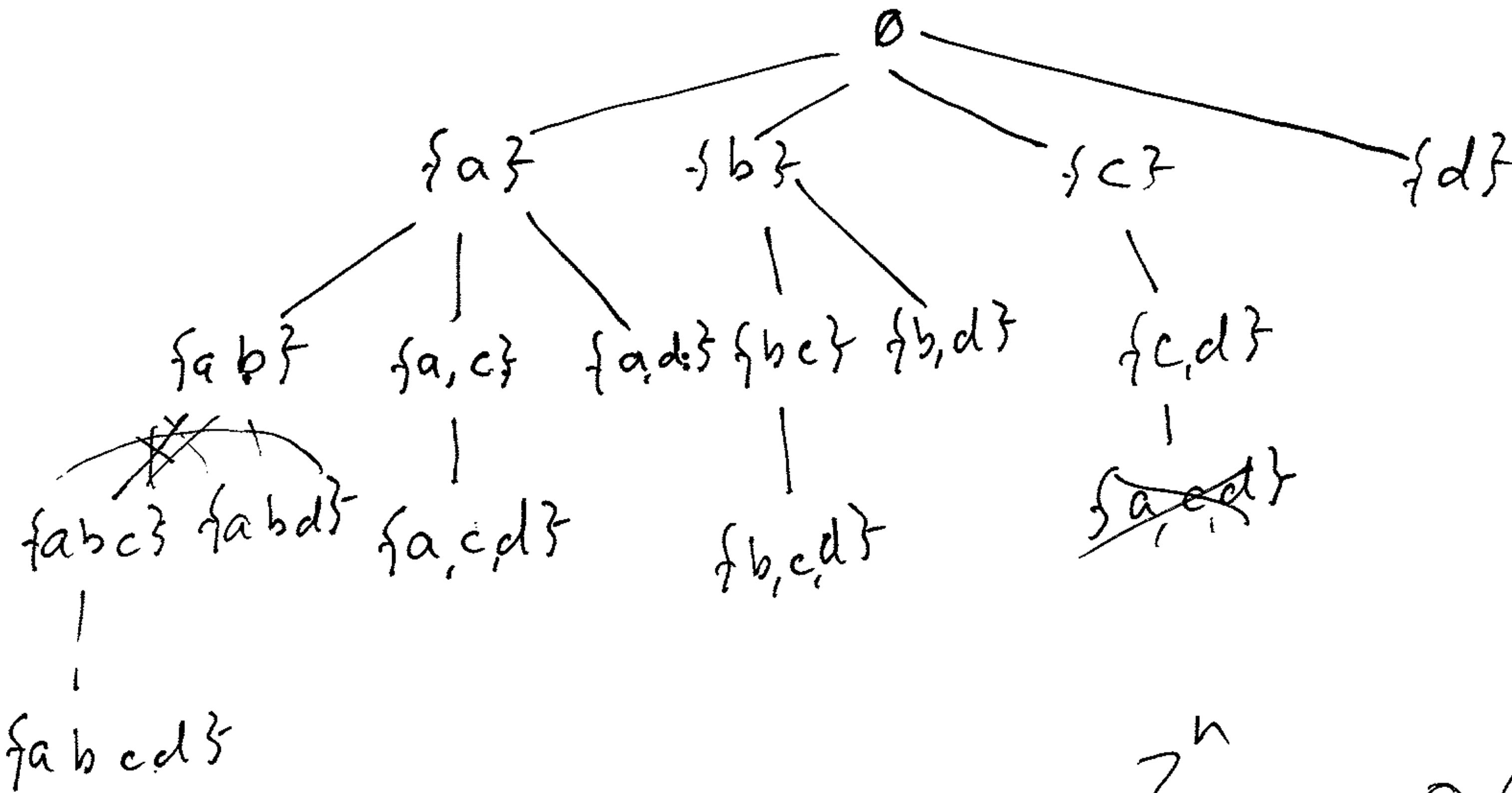
Iterative Solution:

- generate every possible subset 2^n
- for each one:
  <u>if</u> its feasible, compare it to the best found so far, if better update the best
- output the best

$$2^{100}$$



a_2
a_1
a_3
a_4
a_3
a_4
wt = 8



$$\frac{1}{2}\ \frac{2^n}{\cancel{2}} \in O(2^n)$$
...

KNAPSACK(K, j, S)

Input: An instance of the Knapsack problem $K = (A, wt, val, W)$

an index j, a partial solution $S \subseteq A$ consisting

of elements not numbered more than j

Output: ~~An~~ A solution to K, S' that is at least as good as S

initial call:

KNAPSACK(K, 0, $\emptyset$)

```
if (j = n)
  return S
S_best ← S
for k = j+1 ... n
  S' ← S ∪ {a_k}
  if ( wt(S') ≤ W )    // feasible
    T ← KNAPSACK(K, k, S')
    if (val(T) > val(S_best))
      S_best ← T
return S_best
```

Assignment Problem

Given: n people $p_1\ p_2 \cdots p_n$

n tasks $t_1\ t_2 \cdots t_n$

and a cost matrix C_{ij} = cost of assigning person i to task j

Output: An assignment of tasks to people ~~that~~ such that:

- 1 task to 1 person
- minimizes total cost

$O(n^3)$

$O(n!)$

$$\begin{array}{ccc} & & t_1 & t_2 & t_3 \\ \text{people} & \begin{array}{c} p_1 \\ p_2 \\ p_3 \end{array} & \multicolumn{3}{c}{\left[\begin{array}{ccc} 10 & 1 & 3 \\ 8 & 2 & 1 \\ 4 & 2 & 2 \end{array}\right]} \end{array}$$

$p_1 \to t_1$ total: 14

$p_2 \to t_2$ $(t_1\ t_2\ t_3) \to 1\ 2\ 3$

$p_3 \to t_3$

$2\ 3\ 1$

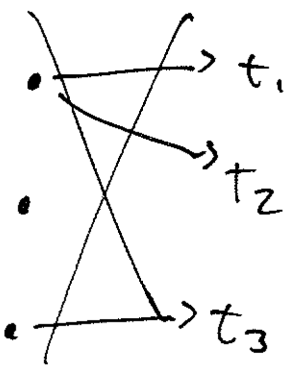
$p_1 \to t_2 \longrightarrow (t_2\ t_3\ t_1)$

total: 6 (better)

permutation

$p_2 \to t_3$

$p_3 \to t_1$



• generate every bijection

Best ?

$P_1 \to t_2$ $(t_2\ t_3\ t_1)$

$P_2 \to t_3$

$P_3 \to t_1$ ~~2 3 1~~