5 Brute-Force

Thursday, February 27, 2025 1:17 PM

- Whose force — Computer's.
- Use the computer to "just go and solve" problem
  - by using the problem's definition directly
  - Usually your first solution.

- PROBLEM : Power

$$a^n = \underbrace{a \cdot a \cdot a \cdot \ldots a}_{n \text{ times}}$$

```
FUNCTION bfPow(a, n)
    r ← 1
    FOR i = 1 TO n DO
        r ← r * a
    RETURN r
```

Analysis

basic operation *

$$M(n) = \sum_{i=1}^{n} 1 = n \quad \in \Theta(n) \text{ linear.}$$

NOTE: - Brute-force can be applied to many problems
- fine for your initial solution
- it almost always yields an inefficient solution.

(Sometimes it is ok: the problem is rare and small)

- PROBLEM : Sorting

DEF: Given a sequence $\langle a_0, a_1, a_2, a_3, \ldots, a_{n-1} \rangle$
find a permutation such that
$\langle a'_0 \leq a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_{n-1} \rangle$

- ALGORITHM : "Selection Sort"

[Repeatedly:
find smallest element, put it at the front

after i iterations:
$\langle a'_0, a'_1, a'_2 \ldots a'_i, a'_{i+1}, a'_{i+2}, \ldots a'_{n-1} \rangle$

after i iterations:

$\langle a'_0, a'_1, a'_2 \ldots a'_i, a'_{i+1}, a'_{i+2}, \ldots, a'_{n-1} \rangle$

sorted | unsorted.

- find smallest element in $i+1 \ldots n-1$
- place it at position $i+1$

then after $i+1$ iterations

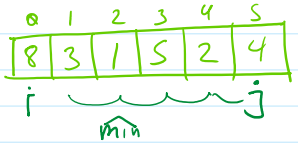
$\langle a'_0, a'_1, a'_2 \ldots a'_i, a'_{i+1}, a'_{i+2}, a'_{i+3}, \ldots a'_{n-1} \rangle$

sorted | unsorted.

Pseudo code

```
FUNCTION SelectionSort(A[0..n-1])
    FOR i ← 0 TO n-2 DO
        min ← i
        FOR j ← i+1 TO n-1 DO
            IF A[j] < A[min] THEN
                min ← j
        SWAP A[i] with A[min]
```

Trace:



n=6

[1 3 8 5 2 4]
i j

[1 2 8 5 3 4]
min
i j

[1 2 3 5 8 4]
min
i

[1 2 3 4 5 8]

https://visualgo.net/en

Analysis:

basic operation <

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} \left((n-1) - (i+1) + 1 \right) = \sum_{i=0}^{n-2} (n-1-i)$$

$$\sum_{i=0}^{n-2} (n-1-i) = \underbrace{(n-1)}_{i=0} + \underbrace{(n-2)}_{i=1} + \underbrace{(n-3)}_{i=2} + \underbrace{(n-4)}_{i=3} + \ldots + \overbrace{\underbrace{(n-1-(n-2))}_{i=n-2}}^{1}$$

$$\sum_{i=0}^{n-2}(n-1-i) = \underbrace{(n-1)}_{i=0} + \underbrace{(n-2)}_{i=1} + \underbrace{(n-3)}_{i=2} + \underbrace{(n-4)}_{i=3} + \ldots + \underbrace{(n-1-(n-2))}_{i=n-2}$$

rewrite:

$$\sum_{k=1}^{n-1} k = \frac{(n-1)\cdot n}{2} = \frac{n^2}{2} - \frac{n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2) \text{ "quadratic"}$$

What about Swap? $S(n) = n-1 \in \Theta(n)$ "linear"

- ALGORITHM : "Bubble Sort"

Intuition: repeat: - Scan array and if two consecutive elements are out of order, swap them.

after i iterations.

$\langle a_0, a_1, a_2, a \ldots\ldots, a_i, a_{i+1}, a_{i+2} \ldots a_{n-1} \rangle$

unsorted. | sorted

In the $i+1$ iteration, the biggest element in $a_0 \ldots a_{i-1}$ is going to "bubble" up to position $i-1$

$\langle a_0, a_1, a_2, a_3 \ldots a_{i-1}, a_i, a_{i+1}, a_{i+2} \ldots a_{n-1} \rangle$

unsorted. | sorted

```
FUNCTION BubbleSort ( A[0..n-1] )
  FOR i ← 0 TO n-1 DO
    FOR j ← 0 TO n-2-i DO
      IF A[j+1] < A[j] THEN
        Swap A[j+1] with A[j]
```

Analysis:

Basic operation: <

$$C(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-1}[(n-2-i)-0+1]$$

$$= \sum_{i=0}^{n-1}(n-1-i) \Rightarrow \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2) \text{ "quadratic"}$$

What about Swap? $S(n) = \frac{(n-1)n}{2} \in \Theta(n^2)$ "quadratic"

- PROBLEM : Search

Given a collection, find if element x is in the collection.

Scenario: collection is an array

- ALGORITHM : Sequential Search

```
FUNCTION SeqSearch ( A[0..n-1], x)
    FOR i<-0 TO n-1 DO
        IF A[i] = x THEN
            RETURN true
    RETURN False.
```

Analysis:

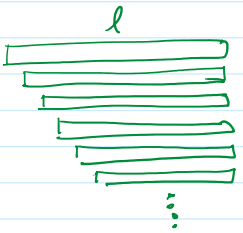
basic operation =

$$C(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$$ "linear"

Python:

```
l.index(e)
[:]
```

```
for i in len(l):
    p = l[i:]
    foo(p)
```



- PROBLEM : String Matching

Given a very large string "text"
and a shorter string "pattern"
find the first occurance of the pattern
in the text.

E.G. text ME_THINK_IS_A_WEASEL
(0 1 2 3 4 5 6 7)
pattern INK

output = 5

Algorithm:

$(t_0, t_1, t_2, t_3, t_4 \ldots t_{n-1})$

$\langle p_0, p_1, p_2 \ldots p_{m-1} \rangle$

Starting at position i=0 to n-m
- compare t_i to p_0, t_{i+1} to p_1, t_{i+2} to p_2 ... t_{i+m-1} to p_{m-1}
- if comparison fails
  - increment i

E.G. text $\overset{0\,1\,2\,3\,4\,5\,6\,7}{\text{ME\_THINK\_IS\_A\_WEASEL}}$
pattern INK

```
FUNCTION StringSearch ( T[0..n-1], P[0..m-1] )
    FOR i ← 0 TO n-m DO
        j ← 0
        WHILE j < m and P[j] = T[i+j] DO
            j ← j+1
        IF j = m THEN
            RETURN i
    RETURN -1
```

Analysis : Parameters: n m $n \gg m$
basic operation $=$

$$C(n,m) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m = ((n-m)-(0)+1)\cdot m$$

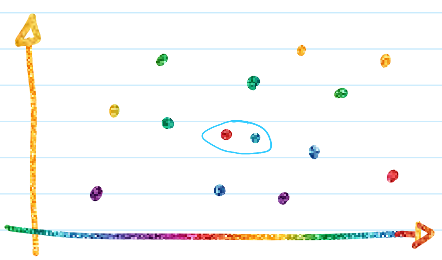
$$= m\cdot n - m^2 + m. \in O(n\cdot m) \quad n \gg m$$

"quadratic"

```
Text = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa............"
Patt = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab"
```

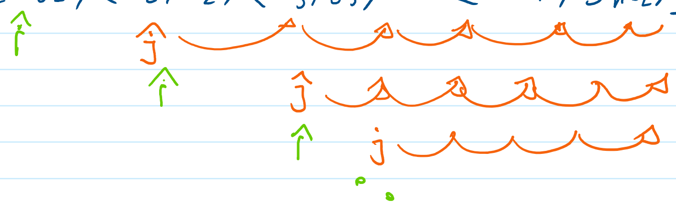
- PROBLEM : Closest Pair

Problem in the 2D Plane.
Given a collection of points $\langle x,y \rangle$ in 2D space
Find the closest 2 points.



$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

input: $p = [\langle x_1, y_1\rangle \langle x_2, y_2\rangle \langle x_3, y_3\rangle \cdots \langle x_{n-1}, y_{n-1}\rangle]$



FUNCTION [illegible] (P[0..n-1])

```
FUNCTION BFClosestPair(P[0..n-1])
    d ← ∞
    FOR i = 0 TO n-2 DO
        FOR j = i+1 TO n-1 DO
            d ← min[d, sqrt((P[i].x - P[j].x)^2 + (P[i].y - P[j].y)^2)]
    RETURN d
```

Analysis:- Parameter n

basic operation ←

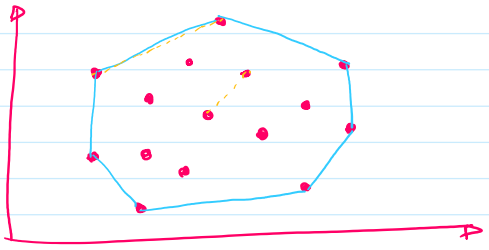
$$A(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} \left((n-1) - (i+1) + 1 \right) = \sum_{i=0}^{n-2} (n-1-i)$$

$(n-1-(n-2))$ $i = n-1$

$$= \sum_{i=0}^{n-2} (n-1-i) = \underset{i=0}{(n-1)} + \underset{i=1}{(n-2)} + \underset{i=2}{(n-3)} + (n-4) + \dots + 3 + 2 + 1$$

$$= \sum_{k=1}^{n-1} k = \frac{(n-1) \cdot n}{2} \approx \frac{1}{2}n^2 - \frac{1}{2}n \in \Theta(n^2)$$ "quadratic"

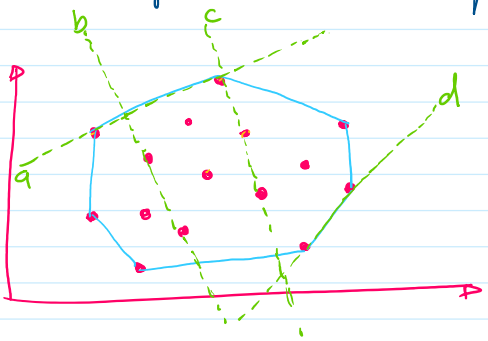
- PROBLEM : Convex Hull



Find the smallest convex poligon that contains all points

Convex poligon:- for any two points p_1, p_2 in the polygon, the line $p_1 - p_2$ resides completely inside the polygon

Note: the vertices of the solution polygon are points in the input.



Pseudocode: Brute_force Convex Hull.

```
for every point p1
    for every point p2, (p2 ≠ p1)
```

```
for every point p1
    for every point p2, (p2 ≠ p1)
        draw the line p1—p2
        for every point p3
            if all p3's are on the same side of p1—p2 then
                p1—p2 is in the solution polygon.
```

Let n be the number of points

then $\Theta(n^3)$ "cubic"

- BRUTE FORCE SRATEGY : EXHAUSTIVE SEARCH

Brute force strategy for the following problem pattern:

- "Given a collection of items of size n find a permutation/combination / subset that:
  - Satisfies some constraints and/or
  - Maximizes / Minimizes a value

S: a collection of items

$|S| = n$

Pseudo code

Generate all candidates
- test for the constraints
- remember the best one

Subsets: 2^n

Permutations: $n!$

Combinations: $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$

- PROBLEM : "Travelling Salesman Problem (TPS)"

"Given a collection of cities find the tour that visits all cities, and returns to the starting point, minimizing cost"

Abstaction:

- Given a weighted connected graph G find a Hamiltonian circuit with minimum weight.

$\langle v_0, v_1, v_2, \ldots v_{k-1}, v_0 \rangle$

by Brute force
find a permutation of $v_1 \ldots v_{k-1}$ that builds a circuit and minimizes cost.

by brute force

Find a permutation of $V_1 \dots V_{k-1}$ that builds a circuit and minimizes cost.

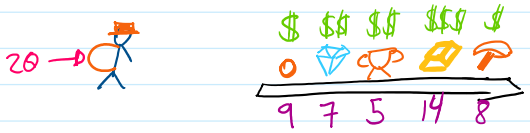
Pseudocode.

```
Brute-force-TPS
    cost ← ∞
    for all permutations p of V1 .. Vk-1
        if cost(p) < cost
            remember p
            cost ← cost(p)
    return p
```

$\approx \Theta(n!)$

Note: no efficient algorithm is known.

- PROBLEM : "Knapsack Problem"



Given a collection of items e_1 e_2 e_3 e_4 ... e_n
each with a weight w_1 w_2 w_3 w_4 ... w_n
and a value v_1 v_2 v_3 v_4 .. v_n

and a "capacity" W

- Find a subset of the items s.t the sum of the weights is less than W that maximizes value.

Pseudocode

```
B.F. KnapSack
    best_value ← 0
    for every subset S of {e1, e2, e3, e4 ... en}
        if the weight of S ≤ W
            if the value of S is greater than best_value
                remember S
                best_value ← value of S
    return S
```

Analysis:

$\approx \Theta(2^n)$

Are ther any better algorithms?

Are ther any better algorithms?

Nobody knows

Knapsack and TSP are NP-hard.
no polynomial algorithm is known.

P = NP?

Common belief = "they don't exist"

- PROBLEM : "Job Assignment Problem"

-Given a list of n workers and a list on n job, and the cost of worker i doing job j

find the assignment of workers to jobs that minimizes cost

e.g.

	Job1	Job2	Job3	Job4
Bob	9	(2)	7	1
Hunter	(6)	4	3	2
Zach	5	8	(1)	2
Justin	7	6	9	(1)

⟨2,1,3,4⟩

Every candidate solution can be represented by a permutation of ⟨1,...,n⟩

BF:
```
best_cost ← ∞
FOR every permutation p of ⟨1,...,n⟩
    IF cost of p < best_cost THEN
        save p
        best_cost ← cost of p
```

Analysis:

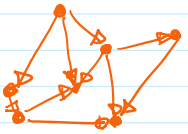
$\approx \Theta(n!)$

Note: Better Algorithms Exist!!

Just because the straight forward brute-force solution is factorial or Exponential (2^n $n!$)
Does not means that better algorithms do not exist.

- EXHAUSTIVE SEARCH on a Graph

$G = \langle V, E \rangle$ V = Vertices, E = Edges



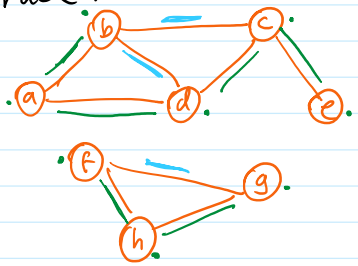
- Consider undirected graphs
- We want to Exhaustively search the graph "visiting" all nodes

- Depth-First Search

```
FUNCTION DFS_main (G)
    mark all nodes as "unvisited"
    count ← 0
    FOR each vertex V in V DO
        IF V is unvisited
            dfs(V)
```

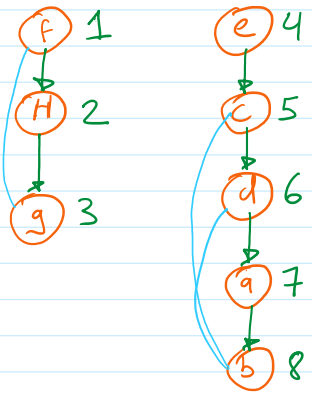
```
FUNCTION dfs(v)
    count ← count + 1
    mark v as visited with count
    FOR each vertex u adjacent to v DO
        IF u is unvisited
            dfs(u)
```

Trace.



DFS

count ~~1 2 3 4 5 6 7 8~~ 1 2 3 4 5 6 7 8



Appications:

- Connectivity
- Cycle checking
- Path existance
- Serialize the graph.

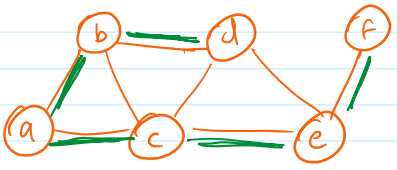
→ Tree-edges of DFS
→ Back-edges

- Breath-First Search

```
FUNCTION BFS_main()
    mark all nodes as "unvisited"
    count ← 0
    FOR each vertex v
        IF v is unvisited
            bfs(v)
```

```
FUNCTION bfs (v)
    VAR Queue.
    count ← count + 1
    mark v with count
    enqueue v
    WHILE queue is not empty
        u ← front of queue
        dequeue
        FOR each vertex w adjacent to u
            IF w is unvisited
                count ← count + 1
                mark w as visited
                enqueue w
```

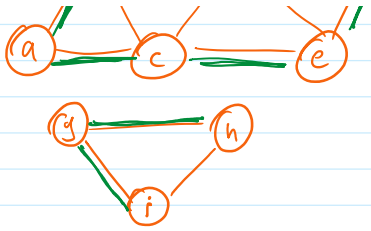
Trace:



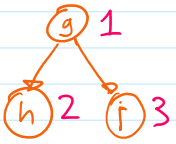
BFS

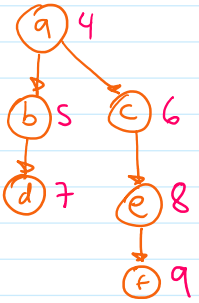
queue

~~a b f d e f~~



mark w as visited
enqueue w





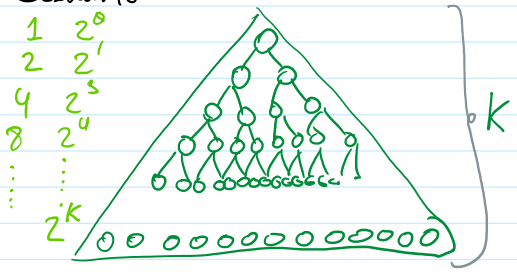
Note: BFS orders the nodes by number of arcs from start node.

Applications:

- Same as DFS.
- Find paths with least arcs

Consider the Memory used.

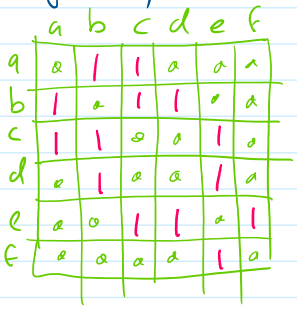
Scenario:

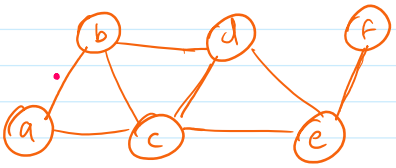


BFS needs a queue of size 2^K

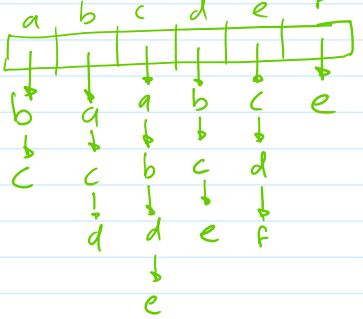
Analysis:

Adjacency Matrix





Adjacency list



$\Theta(|V|^2)$

$\Theta(|V| + |E|)$

EOF