# 5 Brute-Force

Thursday, September 19, 2024 10:55 AM

- 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

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

```
FUNCTION bfPow(a, n)
    r ← 1
    for i = 1 to n do
        r ← r * a
    return r
```

Analysis:

basic operation

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

- Brute force can be applied to many problems but almost always yields an unefficient solution.

  Sometimes it is ok:

  1).- the problem is rare and small

- PROBLEM : Sorting

  Given a sequence $\langle a_0, a_1, a_2, a_3 \ldots a_{n-1} \rangle$

  find a permutation: such that:

  $\langle a'_0 \leq a'_1 \leq a'_2 \leq a'_3 \ldots a'_{n-1} \rangle$

  - Solution : Selection Sort

- Solution : Selection Sort

find the smallest item, put it in the front
find the next smallest item, put it in second place
find the next smallest item, put it in third place
⋮

after $i$ 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

In the $i+1$ iteration

$a'_{i+1}$ is selected from $a_{i+1}, a_{i+2}, \ldots a_{n-1}$

then

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

sorted | unsorted

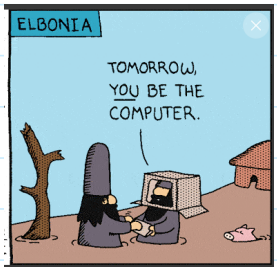
```
FUNCTION SelectionSort(A[0...n-1])
    for i ← 0 to n-2 do
        min ← i
        for j ← i+1 to n-1 do
            if A[j] < A[min] do
                min ← j
        swap A[i] with A[min]
```

Trace:

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 4 | 8 | 9 | 7 | 5 |

i j

min 5




Analysis:

basic operation: <

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

$n-2$ | $1$

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

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

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

What about swap? $S(n) = n-1 \in \Theta(n)$ Linear.

- Solution : Bubble Sort

repeat [ - scan array and if two consecutive elements are out of order, swap them.

after $i$ iterations:

$\langle a_0, a_1, a_2, \dots a'_i, a'_{i+1}, \dots a'_{n-1} \rangle$

unsorted. sorted

In the $i+1$ iteration $a'_{i-1}$ "floats" to its position

$\langle a_0, a_1, a_2 \dots a'_{i-1}, a'_i, a'_{i+1}, \dots, a'_{n-1} \rangle$

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]
```

Trace:

- https://visualgo.net/en

Analysis:

basic operation <

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

$= \sum^{n-1} (n-1-i) \Rightarrow \frac{(n-1)n}{2} \in \Theta(n^2)$ quadratic

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

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

- PROBLEM : Search

  Given a collection: find if element x is in the collection:

  Scenario: collection is an array

  - Solution : Sequential Search

```
FUNCTION SeqSearch ( A[0..n-1], x )
    for i<-0 to n-1
        IF A[i] = x then
            return true
    return false.
```

Analysis:

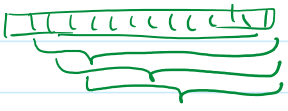
basic operation = =

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

Python:

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

l.index(e)



- PROBLEM : String Matching

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

e.g text ME_THINK_IS_A_WEASEL
pattern "INK"

algorithm:

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

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

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

```
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 return i
    return -1
```

```
          i
text  ME_THINK_IS_A_WEASEL
pattern INK
         INK
          INK
        j INK
            INK
              INK   ✓
```

Analysis:
parameters n m
basic operation =

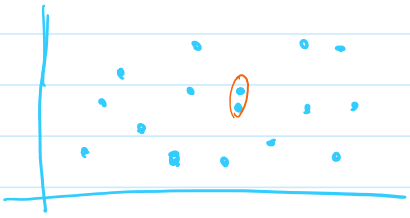
$$C(n,m) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m = (n-m-0+1)m$$

$$= m \cdot n - m^2 + m \in O(n \cdot m) \quad \text{because } n \geq m$$

quadratic.

- PROBLEM : Closest-Pair

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 bf_closest_pair ( 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
```

Trace:

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

i
j
j
j
j

Analysis.

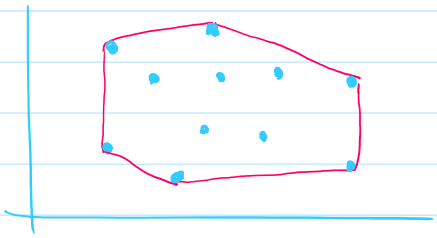
basic operation

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

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

$$= \frac{n(n-1)}{2} \in \Theta(n^2) \quad \text{quadratic.}$$

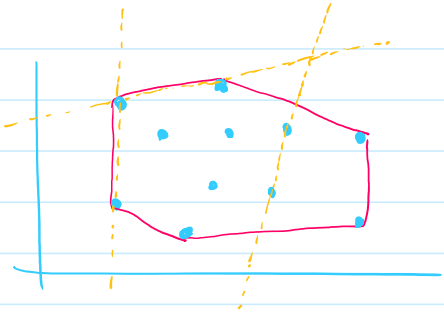
- PROBLEM: Convex Hull



find the smallest convex polygon that contains all points.

convex: for any two points $P_1, P_2$ inside a polygon, the line $P_1 - P_2$ resides completely inside the polygon

- Notice: the convex hull has vertices in the input set.



Pseudocode.

```
for every pair of points p1, p2
    draw the line l = p1 p2
    for every point p3
        if all p3's are on the same side of l
            l belongs to the solution polygon.
```

n: the number of points.

$\in \Theta(n^3)$ cubic

- EXHAUSTIVE SEARCH

Brute force approach to problems of the following type:

- Given a collection of items
  find a permutation / combination / subset
  that
  - Satisfies some constraints

  and/or

- Satisfies some constraints

and/or

- Maximizes or minimizes a value.

$S$ : collection of items

$|S| = n$

permutations: $n!$

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

subsets $2^n$

- **PROBLEM: Traveling Salesman Problem (TPS)**
  - Given a collection of cities, find a tour that visits all cities, and returns to the starting point, minimizing cost
  - Given a weighted connected graph $G$ find a Hamiltonian circuit with minimum weight.

    $\langle v_0, v_1, v_2, v_3, \ldots v_{k-1}, v_0 \rangle$

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

Brute Force Pseudocode.

```
cost ← ∞
for all permutations p of v1 ... vk-1
    if cost(p) < cost
        keep p
        cost ← cost(p)
```

$\approx \Theta(n!)$

- **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" $W$

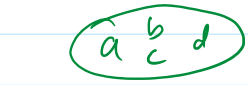
and a "capacity" W

- Find a subset of the items s.t. the sum of the weights of the items is less than W that has maximum sum of values.

B.F.
best_value ← 0
for every subset S of $\{e_1\ e_2\ e_3\ e_4 \ldots e_n\}$
    if weight of S ≤ W then
        if value of S > best_value
            save S
            best_value ← value of S

$\Theta(2^n)$

Why?




| | |
|---|---|
| { } | 0 0 0 0 |
| {a} | 0 0 0 1 |
| {b} | 0 0 1 0 |
| {c} | 0 1 0 0 |
| {d} | 1 0 0 0 |
| {a, b} | 0 0 1 1 |
| {a, c} | 0 1 0 1 |
| ⋮ | ⋮ |

$2^n$

Are there any better algorithms?

Nobody knows

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

Common belief: "they don't exist"

- PROBLEM: Job Assignment Problem
  - Given a list of n workers and a list of n jobs.
  - and the cost of person i doing job j

– and the cost of person i doing job j

find the assignment of workers to jobs that minimizes cost.

eg.

| | Job1 | Job2 | Job3 | Job4 |
|---|---|---|---|---|
| 1 Harsh | 9 | (2) | 7 | 1 |
| 2 Dorian | (6) | 4 | 3 | 2 |
| 3 Rohit | 5 | 8 | (1) | 2 |
| 4 Tung | 7 | 6 | 9 | (1) |

⟨2, 1, 3, 4⟩

Note: cannot assign same job to different workers

Every assignment can be represented as an array
a permutation of the numbers 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
```

$\approx \Theta(n!)$

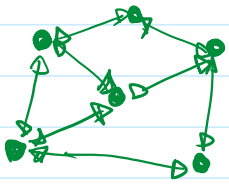
**Note:** Better algorithms do exist!!

Just because a 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.



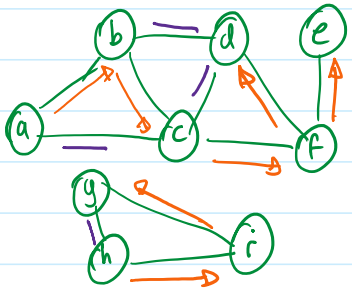
- Consider undirected graphs
- we want to Exhaustively search the graph by "visiting" all nodes.

- we want to Exhaustively search the graph by "visiting" all nodes.
  - Depth-First Search

```
FUNCTION DFS(G)
    mark all vertices as "unvisited"
    count ← 0
    for each vertex v in V do
        if v is unvisited
            dfs(v)
```

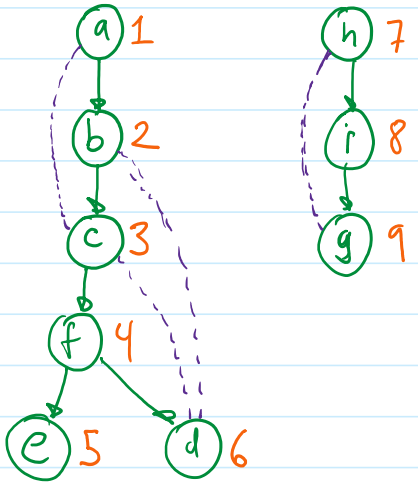
```
FUNCTION dfs(v)
    count ← count + 1
    mark v as visited with count
    for each vertex u adjacent to v do
        if u is unvisited
            dfs(u)
```

Trace.




DFS:

count
1
2 3
4 8
6




— Tree edges of DFS
— back-edges.

Applications:
- Connectivity
- Cycle checking
- path existance
- Serialize the graph

  - Breadth-First Search

Algorithm:

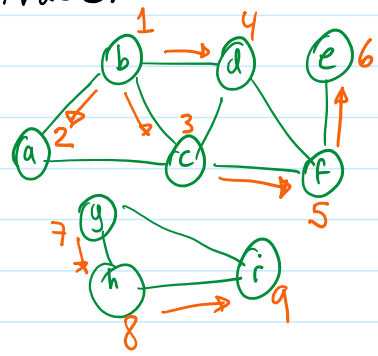
```
FUNCTION BFS
    mark all vertices as "unvisited"
    for each vertex v
        if v is unvisited
```

```
FUNCTION bfs(v)
    VAR Queue.
    count ← count + 1
    mark v with count
```

```
for each vertex v
    if v is unvisited
        bfs(v)
```

```
count ← count+1
mark v with count
enqueue v
while queue is not empty
    y ← front of queue
    dequeue
    for each vertex w adjacent to y
        if w is unmarked
            count ← count+1
            mark w
            enqueue w
```

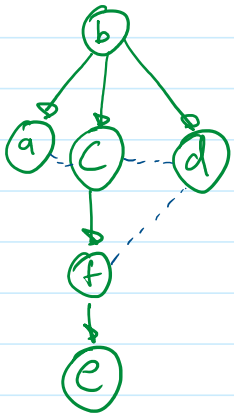
Trace.




BFS

queue

~~b a c d f e g h i~~





Tree edges

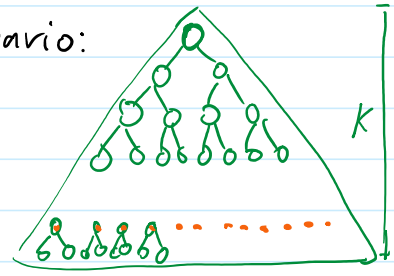
Cross-edges.

Application:

- Same as D.F.S.
- Find paths with the least edges.

Consider the Memory use:

Scenario:




DFS needs a stack of size $k$

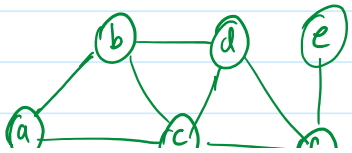
BFS needs a queue of size $2^k$

- Complexity:

The Data structure you use matters.

Adjacency Matrix

| to / a | b | c | d | e | f |
|---|---|---|---|---|---|

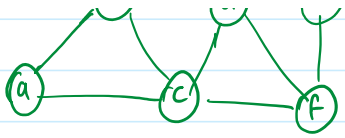



Adjacency List.

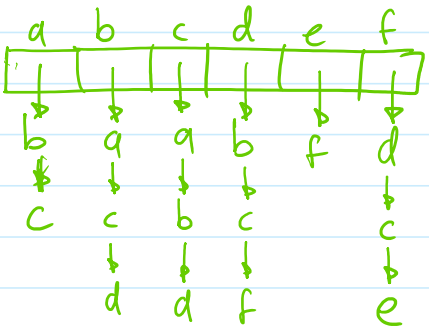
a b c d e f

Adjacency Matrix

| from \ to | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | 0 | 1 | 1 | 0 | 0 | 0 |
| b | 1 | 0 | 1 | 1 | 0 | 0 |
| c | 1 | 1 | 0 | 1 | 0 | 1 |
| d | 0 | 1 | 1 | 0 | 0 | 1 |
| e | 0 | 0 | 0 | 0 | 0 | 1 |
| f | 0 | 0 | 1 | 1 | 1 | 0 |




Adjacency List




$\Theta(|V|^2)$

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

EOF