Date __/__/__

Saathi

Searching

linear search → O(n)

Binary search → O(log(n)) → needs size before
if not known then
taken O(n) time to
find the size first.
→ sorted array needed.

Jump search → m blocks checking

Exponential search - O(log(n)) time
→ find range in the element
→ then perform binary search.

Sorting

Selection sort → Smallest element to first.
Inplace algorithm
$O(n^2)$
O(n) → swaps →
not stable

Bubble sort → Sort neighbours. Worst case (n * n) Best O(n)
good for almost sorted array. In place. Yes stable.

Page No.

Date ___/___/_____

Saathi

Insertion sort: Picks an element and places it in sorted array

Time complexity = $O(n*2)$

space → $O(1)$

minimum when already sorted

max when reverse order.

sorting in place

stable

small no of elements.

Binary insertion sort.

Merge sort → Divide into part then merge.

(n log(n))

O(n) → extra space

~~Bad~~

Not in place

stable algorithm

for linked lists

Quicksort → pick a pivot element partition put all smaller element to left and larger to right of the pivot

$O(n^2)$ worst case
$O(n \log n)$ best case
not stable
In place

Merge sort using linked list no write
$O(n \log n)$
insert any new element easily.

for almost sorted array use insertion sort for $n \log(k)$ complexity.

↓ heap more efficient

quicksort → pick a pivot element partition
put all smaller element to left and
larger to right of the pivot

$O(n^2)$ worst case
$O(n \log n)$ best case
not stable
in place

merge sort using linked list no write
$O(n \log n)$
insert any new element
easily.

for almost sorted array use insertion sort
for $n \log(k)$ complexity.

↓ heap more efficient

Date ___ / ___ / ____

Data structures

Linked list

- Dynamic size
- Ease of insertion/deletion
  - → Random access not allowed.
  - → Extra memory

Class

```
Node {
public:
  int data;
  Node* next;
};
```

Array from stack memory
Linked list from heap.

stack <stack>

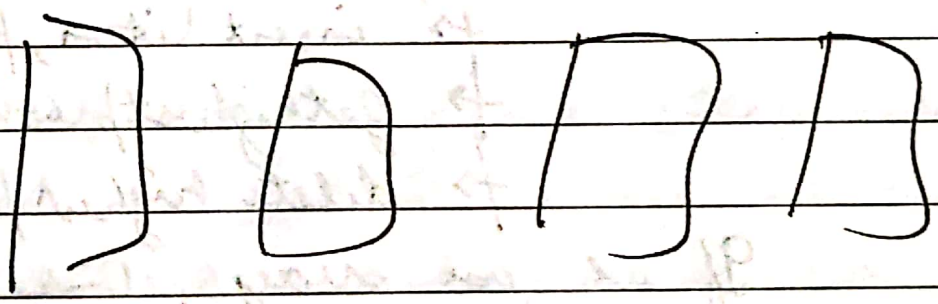
DFS

LIFO

- push ()
- pop ()
- ~~empty~~()
- stop ()

stock span problem → normal method is $O(n^2)$

s[0] ... ... ^



S[n] where no of days the price is less or equal.

to solve this

pop the stack till it is empty or we find an element which is greater than current.

$O(n)$

Queue :

FIFO . BFS

- enqueue
- dequeue

Priority queue

highest priority dequeue first.

- insert (item, priority)
- getHighestPriority ()
- deleteHighestPriority ()

If we use array, linked list → (only last dp. is efficient)

O(1) , O(n)

we use heap (array) generally

O(log n) , O(log n)

if we use fibonacci heap

O(log n) , O(1) , O(log n)

dequeue (double ended queue)

Circular tour that visits all petrol pumps (Circular tour)

→ distance to next petrol & amount of petrol find the first point from where truck can complete all.

If we use brute force we need to check for all petrol pump $O(n^2)$.
but we will use queue.

We add to queue start and add or petrol if we encounter negative petrol we remove start and increase it till the result becomes positive then repeat - If we start the initial start there is no result.
$O(n)$ time complexity.

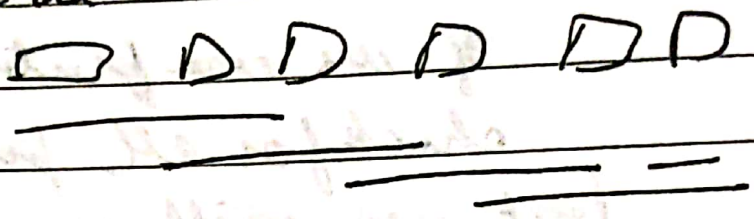
Date ___ / ___ / ____

(Maximum of all subarray of size k)

Maximum of all subarray of size k

find the maximum among the subarray of size k

brute force



then take one subarray second maximum

O(n*k)

using dequeue ~~Stack~~

O(n)

BST O(n log k)

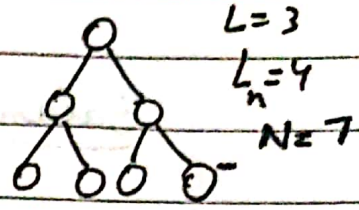
Method to print '1' binary no (n binary)

1) Create an empty queue of string.
2) Enqueue the first binary no ("1")
3) Now run a loop for generating binary
   a) Dequeue and print the front of queue
   b) Append 0 at end of the front and enqueue it
   c) Append 1 and enqueue it

Page No.

Date __/__/____

(Saathi)



Binary tree

→ Access faster than linked list slower than Array and deletion/insertion

→ ~~Nodes~~ No upperlimit on no of elements.

→ level → no of nodes in path from root to node inclusive of both

→ The maximum no of nodes at level 'i' of a binary tree is 2^{i-1}.

→ height → no of ~~edges~~ nodes b/w root and leaf nodes

→ ∴ maximum no of nodes = $2^h - 1$

→ If N nodes minimum height is $\log_2(N+1)$

→ If L leaves then $(\log_2 L)+1$ levels

Full binary tree → Every node has 0 or 2 children

Complete binary tree → All level except last are filled and last has all elements to atmost left as possible.

Perfect binary tree → A binary tree in which all internal nodes have 2 children and all leaves at same level.

Page No.

Balanced binary tree

A binary tree is balanced if the height of the tree is $O(\log n)$ where n is the no of nodes.

→ $L = (K-1) * I + 1$. (K-ary tree)

L = no of leaf nodes

I = no of internal nodes

→ $\sum_{v \in V} \deg(v) = 2|E|$

→ $L = T + 1$

where L = no of leaf nodes

T = no of internal nodes with 2 child

Tree traversals

DFS → inorder (left root right)

preorder (root left right)

postorder (left right root)

BFS →

All traversal require $O(n)$.

for space BFS → $O(w)$ width of tree (>)

$O(h)$

for BFS

1) Create an empty queue q
2) temp = root;
3) Loop while temp != NULL
   → print temp's data
   → add temp children to queue
   → Dequeue a node q and assign its value to temp

time O(n) | O(n) space complexity

Diameter → max distance b/w two leaf passing through root.

→

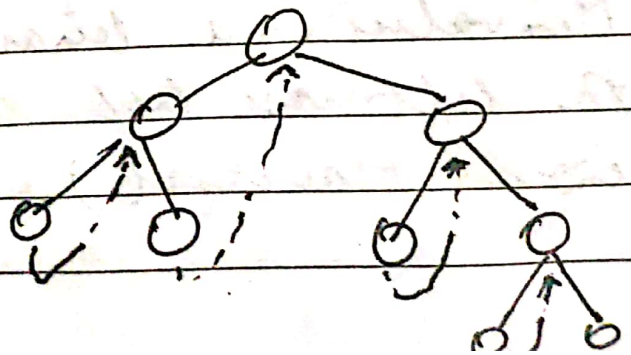
Threaded binary trees

left/right child

The empty pointers of point to the inorder predecessor and successor respectively

in case of double threaded

in single threaded ~~left~~ right child pointers point to successors.



Page No.

Max height of binary tree → O(n)

Binary Search tree:

→ left child only nodes less than the node
→ right child only nodes greater than the node
→

→ Search time complexity is O(h) and
→ insertion is also O(h) = O(log n)

Inorder traversal of bst produces sorted result.

→ Deletion from binary tree is.
  → If it is leaf directly delete
  → If it has only one child delete and move child there
  → If it has two child move the inorder successor there and move
  &
Min value in a binary search tree is the leftmost leaf element (first in the inorder traversal)

Heap

Binary heap

It is a complete tree.
→ It is either min heap or max heap.
It is usually represented as an array.

root → A[0]
for node (i)

(i-1)/2 → parent
(2*i+1) left child
(2i+2) right child

BFS → to achieve array order

→ Heap sort → O(n log n) to sort an array
→ priority queue → insert(), delete(), extractmax(), decrease key in O(log n) time. Binomial/Fibonacci heap are variation of binary heap.

Problems like:
→ K^th largest element
→ sorting.
→ Merge K sorted array
→ Prim's MST, Dijkstra's shortest path

Page No.

Operations

- getMin() → returns the root element of min heap O(1)
- extractMin() → Removes root O(log n)
- decreaseKey() → O(log n) decreases value of key
- insert() → O(log n).
- delete() → O(log n)

To get n^{th} largest element $(n \log n) \times O(n)$

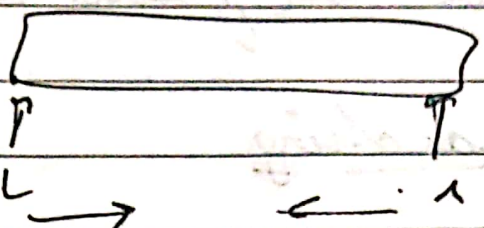
Hashing

Insert data using key O(1)
Get data using key O(1)
delete data using a key O(1)

Hash function: A function that converts a given big phone no to a small practical no.

Hashtable: or array that stores pointers to records corresponding to phone no.

To find sum we need to sort the array



STL library

- Algorithms
- Containers
- functions
- Iterators

Algorithms

- Sorting

sort (start, end)

[start address , end address)

for descending order third argument greater<int>()

for sorting for a struct third argument used which is a comparator which is a boolean function.

Searching

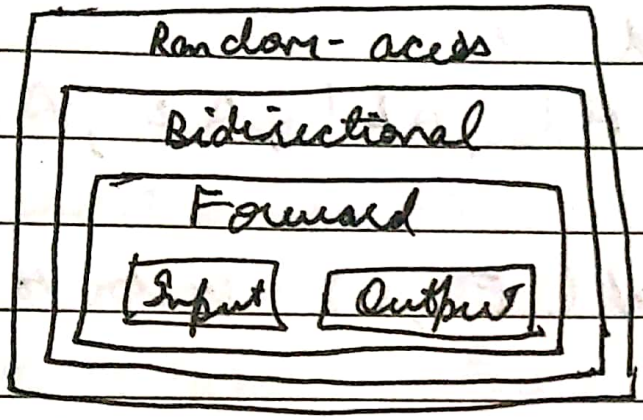
- Binary - search (startaddress, endaddress, value, sorted way)
  - fourth argument is size
  - fifth argument is a comparator

- void * bsearch (const void* key, const void* ptr, std::size_t count, std::size_t size, * comp
- return pointer to the result

- reverse (first_iterator, last_iterator)
- * max_element (first_iterator, last_iterator)
- * min element (first_iterator, last_iterator)
- accumulate (____, ____)
- lower_bound (____, ____, x)
- upper_bound (____, ____, x)
- next_permutation (____, ____)
- prev_permutation (____, ____)
- accumulate (____, ____, initial)

Page No.

Iterator



Vector → RA

list → bidirectional

vector<int>::iterator i;

→ No need to keep track of max elements

→ Dynamically add or remove elements

Date ___ / ___ / ___

useful algos

is_sorted_until ()

is_sorted ()

merge (A1 , AL1 , A2 , AL2, A3)

make_heap (F , L) → create to heap (max)

front()

push_heap ( ) → use push_back before

pop_heap ( ) → use pop_back after

is_heap ( )

is_heap_until ()

max → on list

minmax → pair < ? ;

max_element → on iterator

min_element → on iterator

minmax_element → on iterator

max → on list

Containers

Vector → It is same as dynamic arrays.
- → Data insertion at end.
- → Can be used with iterators.
- → Removing takes constant time
- → Insertion usually takes constant time

Functions on vectors are:-

Iterator
- → begin() → first pointer iterator
- → end() → last
- → rbegin → reverse begin (last)
- → ~~last~~ () → rend
- → cbegin - constant iterator
- → cend → constant iterator

```
auto i = .begin()
```

Capacity:
- → Size() → returns the size
- → max_size() → ~~capacity~~ max element size it can hold
- → capacity() → max element
- → empty() → true/false

Page No.

Element access:

[g] → g^{th} object
at(g) → ↗

front() → reference to first element
back() → last element
data() → reference to internal data

Modifiers:-

assign(X, Y) → if X and Y int then X value to Y elements, if X is array then Y is last count of array.

push_back(value) → pushes value to back
pop_back() → pop's the last value
insert(X, Y) → insert a specified element before the position Y.
insert(position, size, val) → inserts the value size time
insert(position, begin, end) → iterator
clear() → removes all elements O(N)
erase(position) → remove element at the pos
erase(starting position, ending position).
swap(vector2) → swap two vectors (whole) not elements

list

Also some list push_front, pop_front extra

merge()
sort()

Queue

empty()
size()
swap()
emplace()
front()
back()
push(g)
pop()

stack

empty() O(1)
size()
top()
push(g)
pop()

priority_queue (used for heap)

< type, vector type, Comparison Obj >

empty()
size()
top()
push()
pop()
swap()
value_type()

Algorithms

→ To find common element in all 3 arrays

if x < y
then i++
if y < z
then j++
else
z++
if x == y == z
i++, j++, z++

→ Sorted array find a no whose sum is equal to X. (find sum x)

start from i = 0 & j = n
if sum < x increase i
if sum > x increase j.

Fibonacci numbers

$$F_n = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ F_{n-1} + F_{n-2} & 2 > n \end{cases}$$

$F_n \geq 2^{n/2}$ $n \geq 6$

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Normal algorithm

$T(n) = 2$ if $n = 2$

$T(n) = T(n-1) + T(n-2)$

Efficient algorithm

Store F_{n-2} and F_{n-1} and use them to solve F_n iteratively.

Greatest common divisor

Put a/b in simplest form.

$\frac{a/d}{b/d}$.

Need does divide both a and b. d is as large possible

Normal algorithm

Search all no from 1 to a+b to check condition

Efficient algorithm

Let a' be remainder when a is divided by b.

Then

$$gcd(a,b) = gcd(a',b) = gcd(b,a')$$

```
gcd (a, b)
if (b=0)
    return a
a ← a%b
return GCD(b, a)
```

Big O notation

$O(1) < O(\log(n)) < O(\log(n)^2) < O(n^{1/2}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(C^n) < O(n!)$

Sum of n fibonacci no

$S(n) = F(n+2) - 1$ ⇒ $F_{n-1} = F_n - F_{n-2}$

$F_{n-2} = F_{n-1} - F_{n-3}$

Sum of square of fibonacci no

$S(n^2) = f(n) f_{n+1}$

fibonacci no modulo m

→ fibonacci no repeat after certain intervals when modulo is taken the series starts with 01.
find period and reduce the loop.

Greedy algorithm

take the no that has the best outcome in the current situation.

Car fueling problem (Circular tour)

Drive farthest reachable gas station reduce the problem again.

Main ingredients of greedy algorithms

- greedy choice
→ reduction to a subproblem
→ safe move etc.

Strategy

Analyze the problem to have a greedy approach
Then prove this greedy choice is a safe move

Children celebration problem

→ divide children in minimum no of groups
→ age diff of any two kids in a group should not be more than one year

Consider the points on line as age of children. select min no of segments of length 1 that cover all the children

long hike problem

knapsack → maximum weight
every item, weight of item
fit maximum value per unit.

while knapsack is not full
choose item i with max $\frac{v_i}{w_i}$ & fill as much as possible

Greedy algorithm

Used for optimization problems.
or getting approximation solution to NP-hard problems

→ Activity selection problem.

You are given n activities with start and end time
Select the maximum no of activities that a person can perform.
sort by finish time.

→ Kruskal's minimum spanning tree.
O(E log V)
find the edges that connect all vertices of a tree.
V vertices V−1 edges

1) Sort all edges in non decreasing order of their weight
2) Pick the smallest edge. Check if a cycle is formed if no include this edge else discard
3) Repeat 2 till V−1 edges there.

→ find a cycle in a directed graph

O(V+E)

we need to detect a back edge

→ DFS traversal we keep track of vertices in recursion stack.

→ find a cycle in an undirected graph

O(n)

Using disjoint set → O(log n)

↓

data structure used to show disjoint sets.

it can perform two functions union & find.

Huffman coding (priority queue use)

after O(n) (n log n)

encoding of one character is not the prefix of another. such that there is unique decompression.

→ Build huffman code

→ Traverse huffman tree and assign it to

Steps to build a huffman tree

1) Create a leaf node for each unique character and build a min heap • used as priority queue. The least frequent character at root.

2) Extract two nodes with minimum frequency from heap

3) create a new internal node with a frequency equal to the sum of nodes frequency. Make first extracted node as left and second as right. Add this node to the heap

4) Repeat step 2 & 3 - until the heap contains one node. The remaining node is root node the tree is complete

To print code

to move left add 0. to move right add 1

Prim's minimum spanning tree's $O(V^2)$

(for adjacency matrix)

1) Create a set mstSet that keeps track of vertices included in MST
2) Assign a key value to all vertices. All ∞ except Vertices → 0.
3) While MstSet doesn't include all vertices
   a) Pick a vertex u which is not there in mstSet and has minimum key value
   b) Include u to mstSet
   c) Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For all adjacent vertex v. if weight of edge u-v is less than key value of v, update value

(for adjacency list)

Covering segments by points

Sort the segments by end points
select first segments end point as point.
remove all segments covered by the point.
repeat till all segments are covered

Different commands

" as → a_1 + a_2 — a_k such that $a_i \neq a_j$ for all $0 \leq i \neq j \leq k$

it is possible if only if $l >= k$ where l is the least operand we can use.

Dijkstra's Algorithm

→ Create a Min heap of size V where V is the no of vertices in the given graph. Every node of the min heap contains vertex and its distance from source

→ Initialize Minheap with source vertex as root. All other vertex as ∞

→ While Minheap is not empty.

→ Extract the vertex with Minimum distance value node from Minheap. Let the extracted value be u.

→ for every ajacent vertex v of u, check if v is Minheap. If v is in Minheap and distance is more than u − v plus distance of v. Update Distance value of v.

Job sequence problem

→ Sort the jobs in order of their profit
→ select jobs fit then in sequence at best of their deadline if they fit else discard.

Divide and conquer

Breaking one or more problem by splitting the problem recursively solve the program

binary search

polynomial multiplication

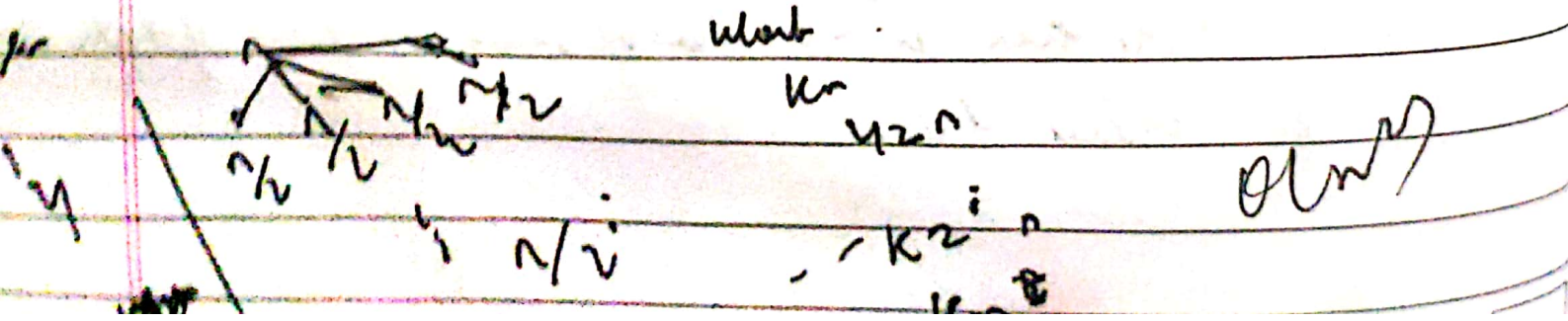
naive algorithm :> Break polynomial as

$A(x) = A_0(x)x^{n/2} + A_1(x)$

$B(x) = B_0(x)x^{n/2} + B_1(x)$

and multiply them we get

$A_0(x)B_0(x)$



Karatsuba approach

$A(x) = a_1 x + a_0$

$A(x) = b_1 x + b_0$

$C(x) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1) x + a_0 b_0$

$C(x) = a_1 b_1 x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0) x + a_0 b_0$

$A(x) = 4x^3 + 3x^2 + 2x + 1$

$B(x) = x^3 + 2x^2 + 3x + 4$

$D_1(x) = 4x + 3$ $\quad D_0(x) = 2x + 1$

$E_1(x) = x + 2$ $\quad E_0(x) = 3x + 4$

$D_1 E_1 = 4x^2 + 11x + 6$ $\quad D_0 E_0 = 6x^2 + 11x + 4$

$(D_1 + D_0)(E_1 + E_2) = (6x + 4)(4x + 6) = 24x^2 + 52x + 24$

② $AB = (4x^2 + 11x + 6)x^4 + (24x^2 + 52x + 24 - 4x^2 + 11x + 6 - 6x^2 - 11x - 4) + (6x^2 + 11x + 4)$.

$= 4x^6 + 11x^5 + 20x^4 + 30x + 20x + 11x + 4$

$O(n^{1.59})$

Master theorem

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ - $(a \geq 0, b > 0, d \geq 0)$

$T(n) = O(n^d)$ if $d > \log_b a$

$= O(n^a \log n)$ if $d = \log_b a$

$= O(n^{\log_b a})$ if $d < \log_b a$

Closest pair problem

- Partition the subset in two using x coordinate mid point.
- Call for subset.
- Find minimum distance b/w the points in subset by using points only inside the mid range.

points and segment

sort the segment by ~~start~~ find the first element that is equal to or less than the key. (i) If equal then take first

sort the segment by end ~~st~~ find the first

element that is equal to or less than key. take that j. no of segment is i - j.

Majority element

if n/2 <= 1 return element.

find mid, split by mid

find the majority element in both

if same for both then return

else count occurances and return.

find inversions (merged of sorted array)

split the set in n/2

get inversions in both.

merge both sets :

- merge is same as in merge sort.

we count the no of inversions. and return

Dynamic programming

Recursion grows a lot. But recursions of some element is used many time. So we store values of certain values by filling the matrix. from small to large.

```
DPchange (money, coins)
MinNumCoins(0) ← 0
for m from 1 to money:
    MinNumCoins(m) ← ∞
    for i from 1 to |coins|:
        if m ≥ coin_i:
            NumCoins ← MinNumCoins(m - coin_i)
            if NumCoins < MinNumCoins(m):
                MinNumCoins(m) = NumCoins
return MinNumCoins(money)
```

Greedy approach fails sometimes because if we take higher amount the remaining might take more input.

Edit distance problem (minimum no of operations to transform a string to another).

$A[1 \ldots n]$ & $B[1 \ldots m]$ what is optimum alignment of an i prefix $A[1 \ldots i]$ of the first string and a j prefix $B[1 \ldots j]$ of the second string last column is either.

an insertion / deletion	mismatch / match	Common Subsequence
$A[1 \ldots i] \quad -$ $B[1 \ldots j-1] \quad B[j]$	$+1$	Matches in an alignment of two string
$A[1 \ldots i-1] \quad A[i]$ $B[1 \ldots j] \quad -$	$+1$	Longest common subsequence
$A[i] \neq B[j]$ $A[1 \ldots i-1] \quad A[i]$ $B[1 \ldots j-1] \quad B[j]$	$+1$	Maximum $\mu=0, \sigma=0$ cridel
$A[i] = B[j]$ $A[1 \ldots i-1] \quad A[i]$ $B[1 \ldots j-1] \quad B[j]$		

Alignment game → Remove all symbol from two strings such that your point are maximized.

score (insertion is - in first
deletion is - in second)

Alignment is a two row matrix

same as edit distance

if we remove last column we are left with optimal alignment of prefixes

let $D(i,j)$ be the edit distance of an i prefix $A[1 \ldots i]$ and j - prefix $B[1 \ldots j]$

$$D(i,j) = \begin{cases} D(i, j-1) + 1 & \\ D(i-1, j) + 1 & \\ D(i-1, j-1) + 1 & A[i] \neq B[j] \\ D(i-1, j-1) & A[i] = B[j] \end{cases}$$

		j	D	I	S	T	A	N	C	E
		0	1	2	3	4	5	6	7	8
i	0	0	1	2	3	4	5	6	7	8
E	1	1	1	2	3	4	5	6	7	7
D	2	2	1	2	3	4	5	6	7	8
I	3	3	2	2	2	3	4	5	6	7
T	4	4	3	2	3	3	3	4	5	6
I	5	5	4	3	3	3	4	4	5	6
N	6	6	5	4	4	4	4	3	4	5
G	7	7	6	5	5	5	5	4	4	5

How to construct alignment

E	-	D	I	T	I	N	G
D	I	S	-	T	A	N	C	E

E	D	I	-	T	I	N	-	G
-	D	I	S	T	A	N	C	E

backtrack from last to start.

Knapsack problem

$$Value(W) = \max_{w_i \in W} (value(W - w_i) + v_i)$$

Fractional knapsack → take fractional/greedy wise

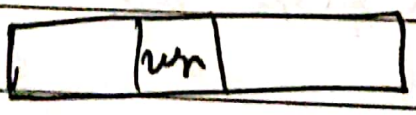
discrete → take whole or next
→ with repetitions or not

with repetitions

$$Value(W) = \max_{w_i \in W} (value(W - w_i) + v_i)$$

without repetitions

single copy of each copy is given then.



$w_{1 \ldots n}$

$w - w_n$

with items

$(\ldots n$

$$Value(w) = \begin{cases} \max\{ value(w - w_i, i-1) + v_i \\ \quad value(w, i-1) \} \end{cases}$$

initialize all value$(0, i) \leftarrow 0$

initialize all value$(w, 0) \leftarrow 0$

for i from 1 to n

  for j from 1 to w

    value$[w][i] \leftarrow$ value$(w, i-1)$

    if $w_i \leq w$:

      val $\leftarrow$ value$(w - w_i, i-1) + v_i$

      if value$(w, i) <$ val:

        value$[w][i] \leftarrow$ val

return value(w, n).

Applying parenthesis to an expression