Greedy Algorithms

Introduction

- Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the aim of finding a global optimum.
- These algorithms are simple, efficient, and often used for optimization problems where we seek the best solution from a set of possibilities.

Key Characteristics

- Greedy algorithms make decisions based on the current best option without considering the overall future consequences.
- They are greedy in the sense that they always choose the option that appears to be the best at the moment.
- Greedy algorithms do not always guarantee the globally optimal solution, but they often produce acceptable results quickly.

Basic Workflow

1. **Initialization:** Start with an empty solution or a solution with the initial state.
2. **Selection:** Choose the best available option based on a specific criteria.
3. **Feasibility Check:** Verify if the chosen option satisfies all constraints and requirements.
4. **Update:** Modify the solution and update the problem state.
5. **Termination:** Repeat steps 2 to 4 until a certain condition is met.

Kadane’s Algorithm

- **Largest Sum Contiguous Subarray**
- Given an array **arr[]** of size **N**. The task is to find the sum of the contiguous subarray within a **arr[]** with the largest sum.

Pseudocode:

- *Initialize:*
  max_so_far = INT_MIN
  max_ending_here = 0
- *Loop for each element of the array*
- *(a) max_ending_here = max_ending_here + a[i]*
  (b) if(max_so_far < max_ending_here)
  max_so_far = max_ending_here
  (c) if(max_ending_here < 0)
  max_ending_here = 0
  return max_so_far

Code:

```
int maxSubArraySum(int a[], int size)
{
   int max_so_far = INT_MIN, max_ending_here = 0;

   for (int i = 0; i < size; i++) {
      max_ending_here = max_ending_here + a[i];
      if (max_so_far < max_ending_here)
         max_so_far = max_ending_here;

      if (max_ending_here < 0)
         max_ending_here = 0;
   }
   return max_so_far;
}
```

Jump Game

- You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.
- Return true if you can reach the last index, or false otherwise.

Solution

```
bool canJump(vector<int>& nums) {
    int canJumpTill = 0;

    for(int i=0; i<nums.size(); i++)
    {
        if(canJumpTill >= i)
            canJumpTill = max(canJumpTill, i+nums[i]);
        else
            break;
    }

    return (canJumpTill >= nums.size()-1);
}
```

Find Original Array From Doubled Array

- An integer array original is transformed into a doubled array changed by appending twice the value of every element in original, and then randomly shuffling the resulting array.

- Given an array changed, return original if changed is a doubled array. If changed is not a doubled array, return an empty array. The elements in original may be returned in any order.

Solution

```
multiset<int> mst;
 for(auto e: a)
     mst.insert(e);
 vector<int> ans;
 while(mst.size())
 {
     int x = *mst.begin();
     ans.push_back(x);
     mst.erase(mst.find(x));
     mst.erase(mst.find(2*x));
 }
 cout<<ans<<"\n";
```

Advantages

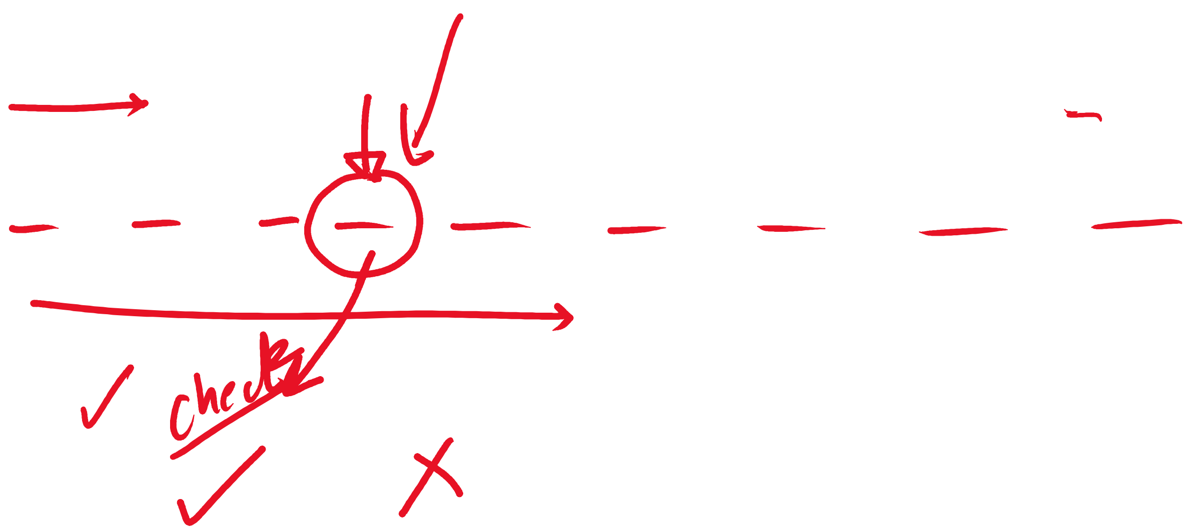
- Greedy algorithms are relatively easy to implement and efficient in terms of runtime complexity.
- They work well for problems where making the locally optimal choice at each step leads to a globally optimal solution.
- They can provide quick solutions for large-scale problems when other methods might be computationally expensive.

Limitations

- Greedy algorithms might not always produce the best solution, as they lack global awareness.
- There's a risk of getting stuck in a suboptimal solution, especially if the locally optimal choices accumulate.
- Careful analysis is required to ensure that greedy algorithms indeed lead to optimal or acceptable solutions.

Thank You

20 June 2023 18:00



$N \quad 10^6$

dp

bruteforce

Greedy

$N \to$ value

$N \to$ weight

N $\to 2^N$

value
x
wt

x
wt

weight $\underline{2}$

10^6

Wt

$\underline{N}$ $\underline{M}$

total

optimal

70%

10^3

x y^m 10^3

10^8

10^6

dP ✓

10^3

$x \times y \approx \underline{10^6}$

$O(n)$

Case 1

i

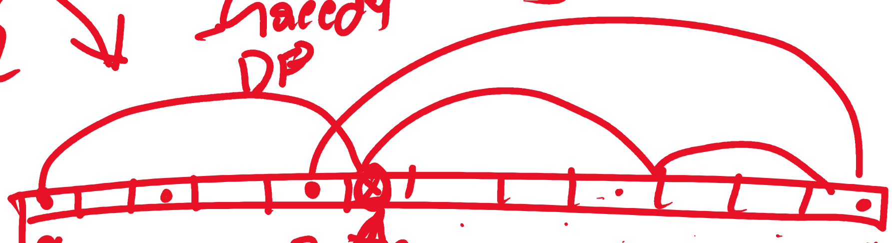
n 10^3

x

$\approx 10^6$

$O(n)$

Greedy
DP



A[1]=2

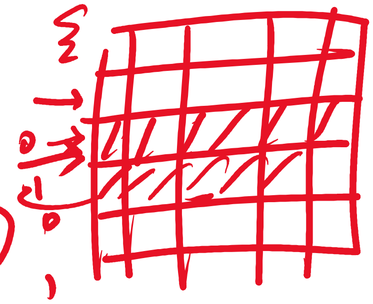
7 4 3

2

$O(N)$

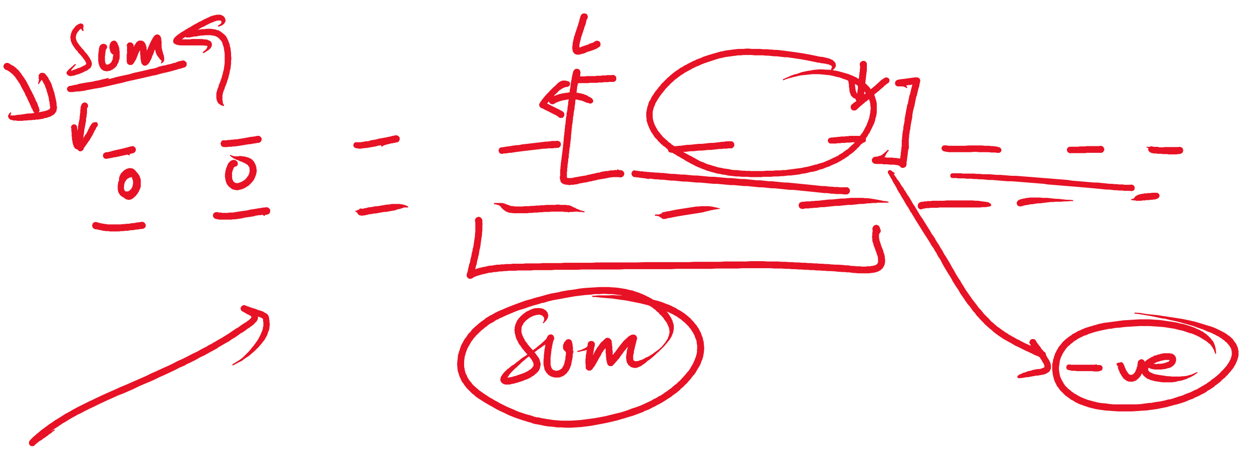
dp[n][m]

i-1 j



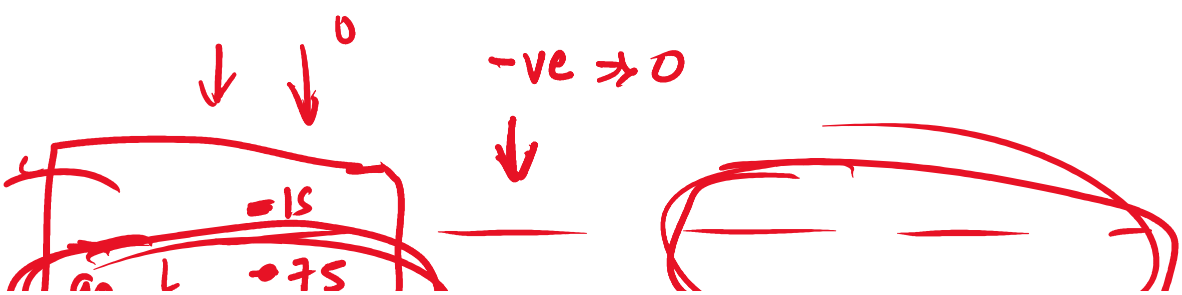
d[i][j]

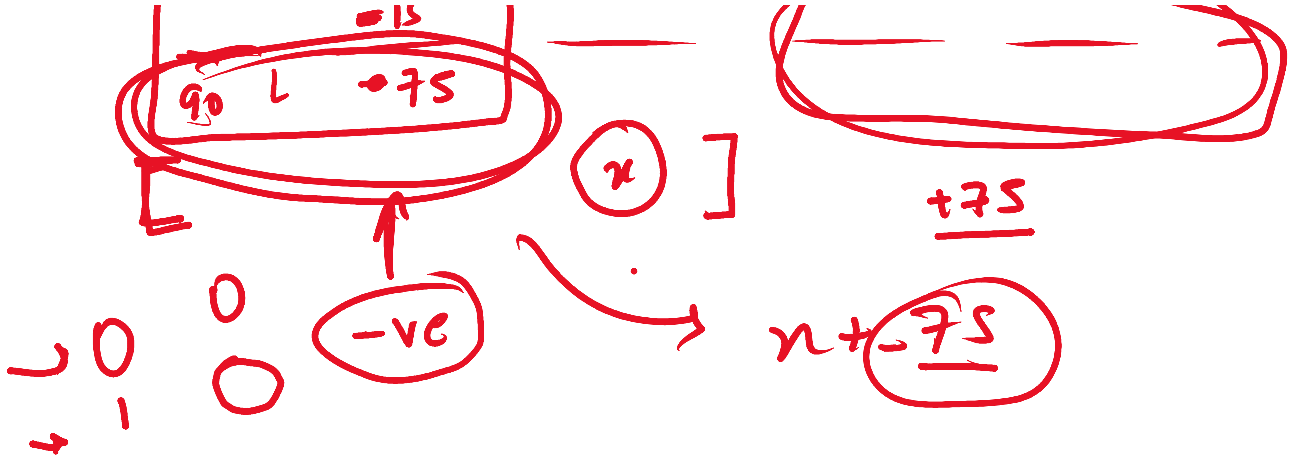
dp[2][m] 10^8



ans = max(ans, sum)

sum < 0
 sum = 0





$dp[2][n]$

Init:

① sel → sum += a[i]

feas → ans = max(ans, sum)

④ updat → sum=0 (sum<0)

Termin

ans

min

DP n^2

vis

n

1 0   0   1 0   0   1 0   0   0

max ind

2   2   0   3

n^2

0	1	2	3
2	2	0	3

$i + arr[i]$

i + arr[i]

max = 2

2 2

5 10 0 0 0 0 0 0 1

+1 +1 -1

5 0 0 0 1 0 0 0 0 10 1

+1 -1

3	y	2	α	β	γ	δ
0	1	2	3	4	5	6

max index Reached $\geq i$

$\geq$ ctur false

$(0 + x)$ → $\boxed{i + a[i]}$

max index = ~~0~~ 3

3 2 1 0 4

$0 + 3 = 3$

0+3

max $\geq i$

max $\geq i$

1+2 = 3

2+1 = 3

Ini → max index = 0

sel → $i + a[i]$

feas check → $i < \text{max index}$

update → max index = max (max,

feas

update → max index = max(max, i + a[i])

Term

$\geq n$

1 2 2 3 4 6 10 20

Original

$[\alpha \quad \beta \quad \gamma \quad \delta]$

$[\alpha \quad \beta \quad \gamma \quad \delta \quad 2\alpha \quad 2\beta \quad 2\gamma \quad 28]$

2 10 1 3 4 20 2 6

2 10 1 3

α

2 10 1 3

2 10 1 3 4 20 26

1 2 2 3 4 6 10 20

x

multiset

smallest

ans

x

$2x$

$2x$

+ve

−ve

m

multiset.erase($2x$)

multiset.erase(multiset.find($2x$))