Exhaustion, branch & bound

Jason

Exhaustion

- Enumerate all possible solutions
- Check whether each of them satisfies the problem's statement
- Find the best one among them

When to use

- When the problem is proved to be difficult to solve
- When the number of possible solutions is small enough
- When the constraints are small (e.g. 1 <= N <= 10)
- When you don't know how to solve the problem quickly

Example 1

- Given a list of integers and an integer M
- Choose a pair of integers such that the sum of them is equal to M
- E.g. A = {1,2,4,8,16}, M = 10 -> Choose 2 and 8

Example 1

- The number of possible solutions is small
  - Number of pairs = N(N-1)/2, where N is number of integers
- We can try to form all pairs and check whether they satisfy the requirement

```
For i = 1 to N-1
        For j = i+1 to N
                if A[i] + A[j] = M then
                        return {A[i], A[j]}
```

Example 2

- Given a list of positive integers and an integer M
- Find a subset such that the sum of the integers is equal to M
- E.g. A = {2,9,15,16}, M = 27 -> Output {2, 9, 16}

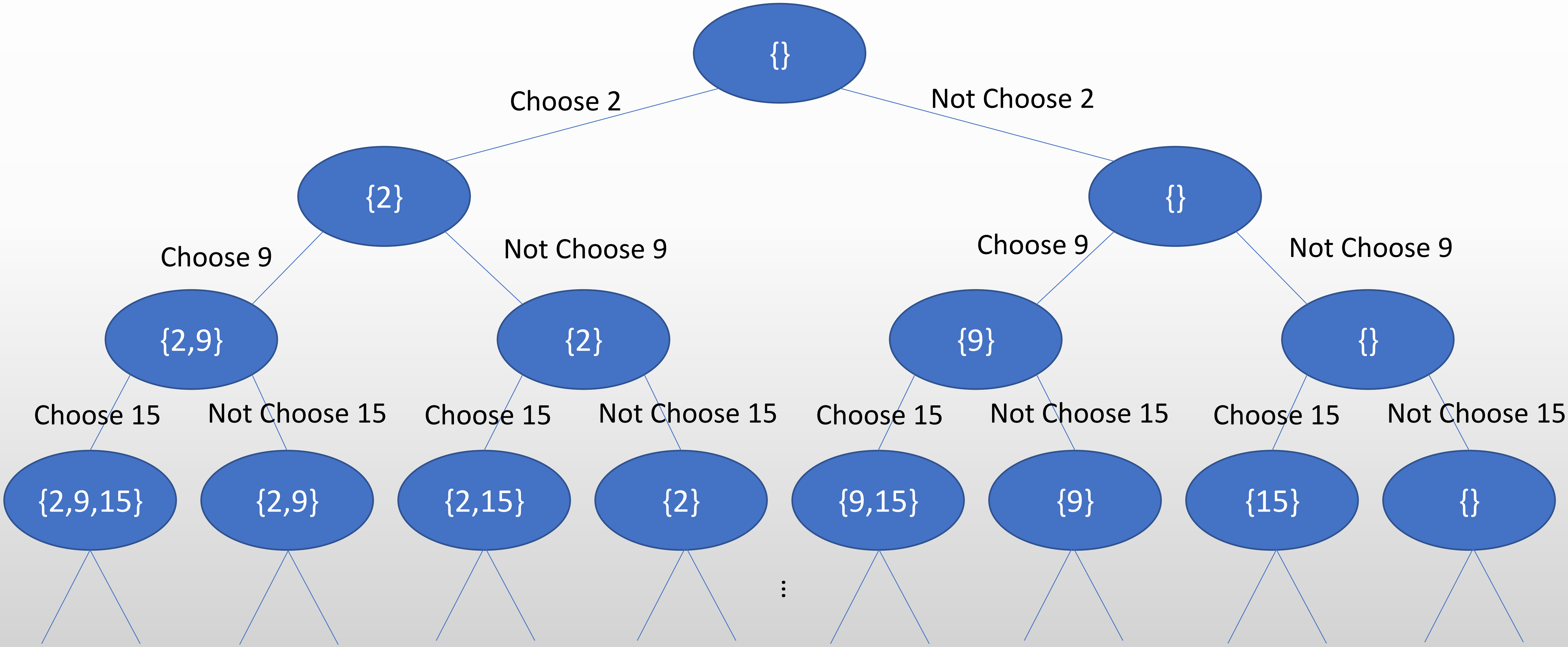
Example 2

- Number of possible solutions = 2^N
- {}
- {2}, {9}, {15}, {16}
- {2, 9}, {2, 15}, {2, 16}, {9, 15}, {9, 16}, {15, 16}
- {2, 9, 15}, {2, 9, 16}, {2, 15, 16}, {9, 15, 16}
- {2, 9, 15, 16}
- When N is small, we can just check all of them

Example 2

- Method in example 1 can not solve the problem
- 1 for loop to choose 1 integer, 2 for loops to choose 2 integers, 3 for loops to choose 3 integers, …
- Use recursion

Example 2



{}
Choose 2
Not Choose 2
{2}
{}
Choose 9
Not Choose 9
Choose 9
Not Choose 9
{2,9}
{2}
{9}
{}
Choose 15
Not Choose 15
Choose 15
Not Choose 15
Choose 15
Not Choose 15
Choose 15
Not Choose 15
{2,9,15}
{2,9}
{2,15}
{2}
{9,15}
{9}
{15}
{}
⋮

Example 2 – Pseudocode

```
Procedure exhaustion(int x, int sum)
    If x <= N then
        choose[x] = true
        exhaustion(x + 1, sum + A[x])
        choose[x] = false
        exhaustion(x + 1, sum)
    Else
        if sum = M then
                output the numbers that are chosen
```

Example 2

- Time complexity: $O(2^N)$

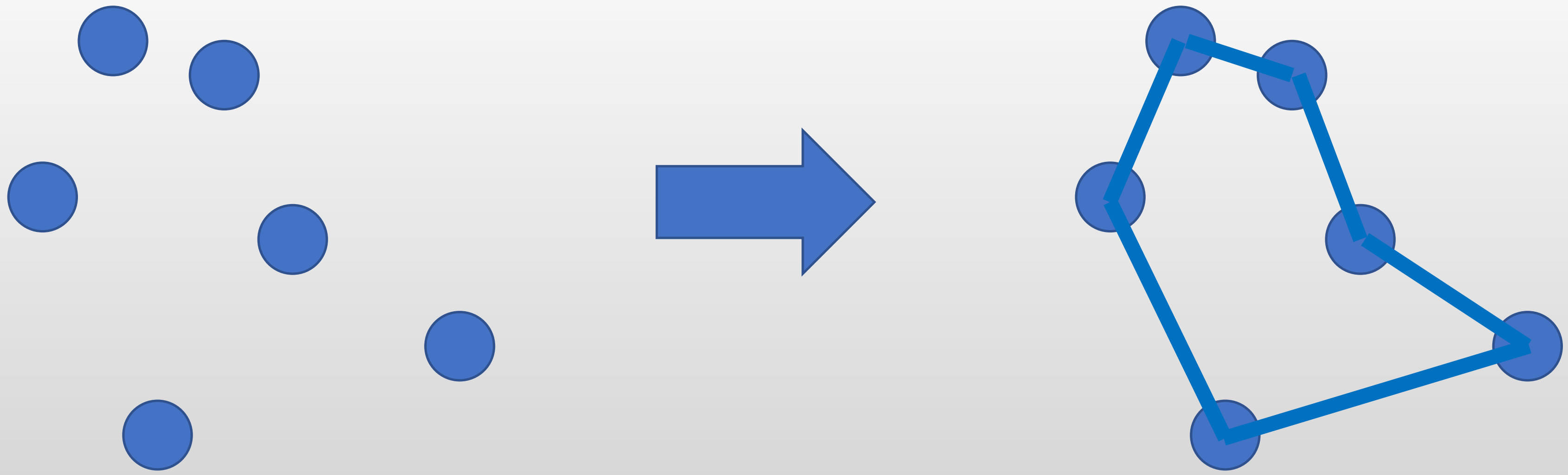
- There exists other more efficient solutions
- Exhaustion is enough to solve the problem if N is small (e.g. N <= 20)

Example

- HKOJ S153 Secret Message

Example 3

- Given the coordinates of N points
- Find out the shortest possible route that visits each point exactly once and returns to the origin point
- Example:



Example 3

- The famous “Travelling salesman problem”
- Proven to be an NP-hard problem
- Cannot be solved in polynomial time (for now)

- The number of possible routes is N!
- Enumerate all of them to find the shortest one

Example 3 – Pseudocode 1

```
Procedure exhaustion(int x)
    If x <= N then
        for i from 1 to N
                if the i-th point is not chosen
                        mark the i-th point as the x-th point of the route
                        exhaustion(x + 1)
    Else
        Calculate the route
        Check if it is the shortest one
```

Time complexity: O(N*N!)

Example 3 – Pseudocode 2

P[i] denotes the i-th point

```
Procedure exhaustion(int x, double total)
    If x <= N then
        for i from x to N
            swap P[x] and P[i]
            exhaustion(x + 1, total + distance between P[x] and P[x-1])
    Else
        total = total + distance between P[N] and P[1]
        Check if it is the shortest one
```

Time complexity: O(N!)

Example 4 – Distinct Permutations

- HKOJ 01031
- Similar to the previous problem
- Given a string of alphabets
- Generate DISTINCT permutations

Example 4 – Distinct Permutations

- Methods from example 3 may fail
- If the input is ACBA, 2 AABC may be generated
- Because 2 ‘A’s are treated as different elements

Example 4 – Distinct Permutations

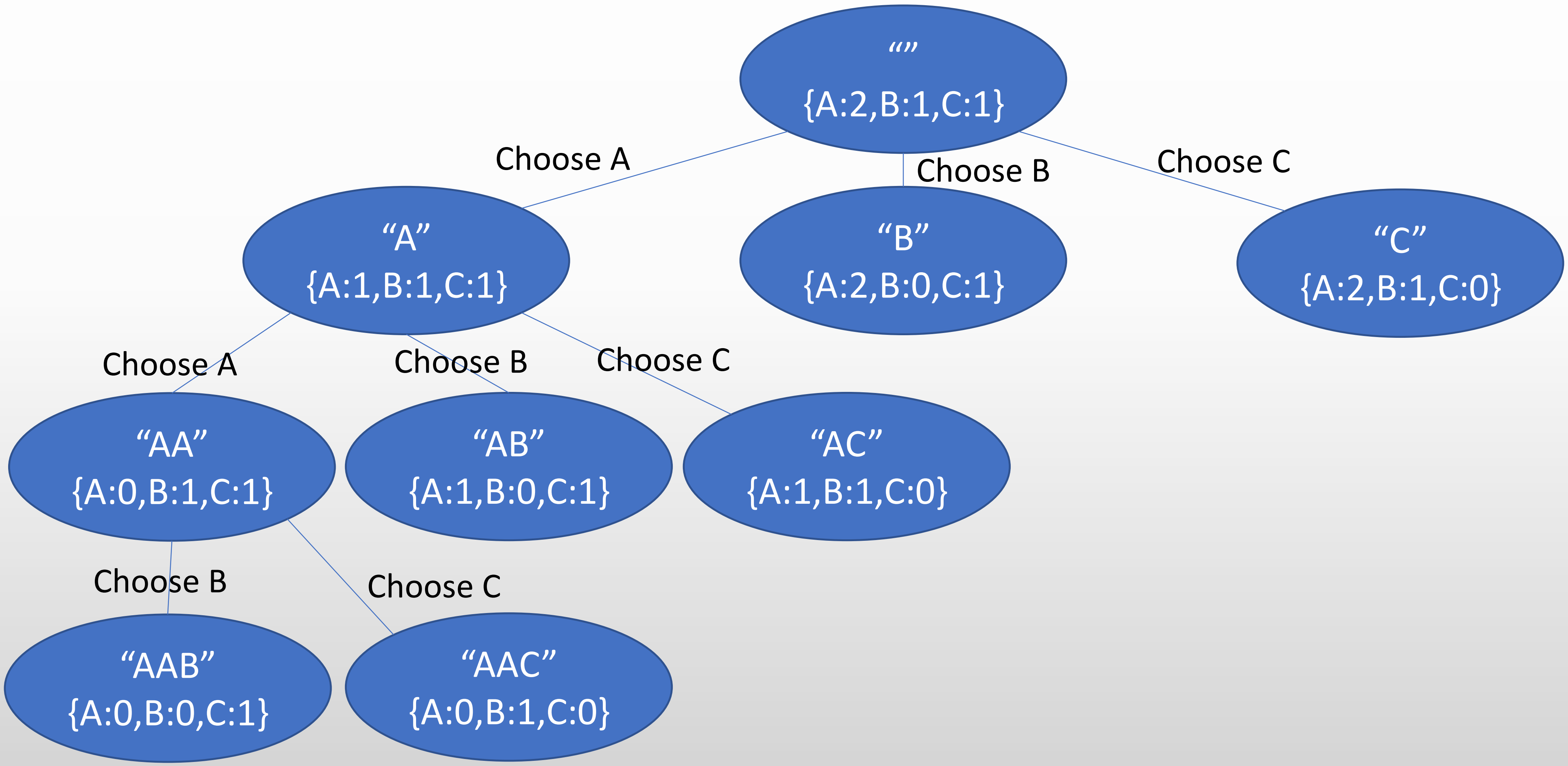
- Solution 1:
- After generate the permutations
- Delete the repeated ones by sorting (or map in c++)

- Okay for this problem, but more time and memory is wasted

Example 4 – Distinct Permutations

- Solution 2:
- Count the frequency of each alphabet
- During the recursion, choose an alphabet that still has quotas

Example 4 – Distinct Permutations



Example 4 – Pseudocode

Let s[1..K] be the output string

Procedure exhaustion(int x)

```
If x <= K then
    for i from 'A' to 'Z'
        if frequency[i] > 0 then
            frequency[i] = frequency[i] – 1
            s[x] = i
            exhaustion(x + 1)
            frequency[i] = frequency[i] + 1
Else
    print s
```

Example

- HKOJ T112 Tetrisudoku

Practice problem

- HKOJ 01014 Stamps
- HKOJ 01031 Permutations
- HKOJ 01035 Combinations
- HKOJ 20296 Safecracker
- UVA 725 Division

Exhaustion

- Sometimes the way we do exhaustion may affect the efficiency
- A part of the answer may determine the rest
- Smaller part is unknown -> less states to be searched

Example 5 – Toggle

- A board with N×N cells
- Each cell is colored either black or white initially
- In each move, you can either 'toggle' a row or a column so that the color of the cells on the entire row/column changes. If the color of a cell is black, it becomes white after it is toggled, and vice versa.
- Finds a sequence of moves that maximizes the number of white cells.

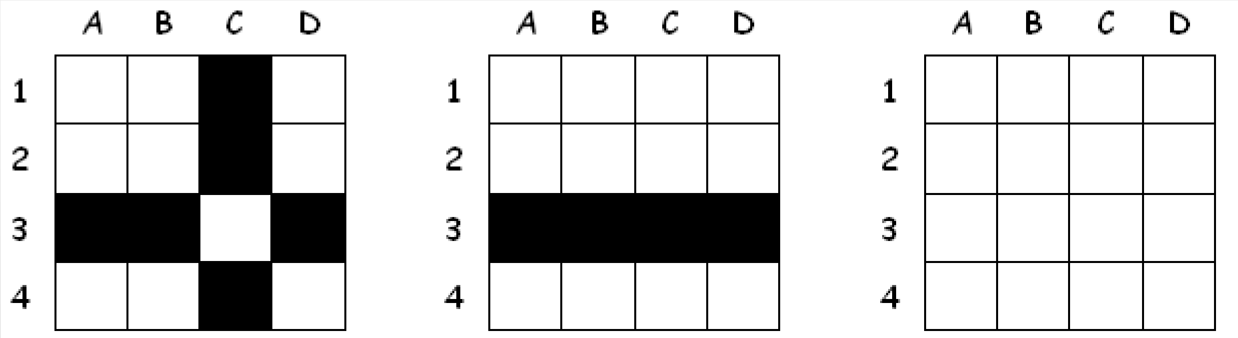


Figure 1 Figure 2 Figure 3

Example 5 – Toggle

- The order of row/column to be toggled does not matter
- Each row/column will be toggled at most once
- Try toggling all combinations of rows and columns
- See which one gives the most number of white cells

Example 5 – Toggle

- Number of combinations = $2^{2N} = 4^N$
- Time complexity : $O(N^2*4^N)$

- Works when N <= 4
- Too slow when N = 16

Example 5 – Toggle

- If the combination of rows to be toggled is fixed, we don’t need to try toggling all combinations of columns
- If a column has more white cells than black cells, we should not toggle it
- If a column has more black cells than white cells, we must toggle it

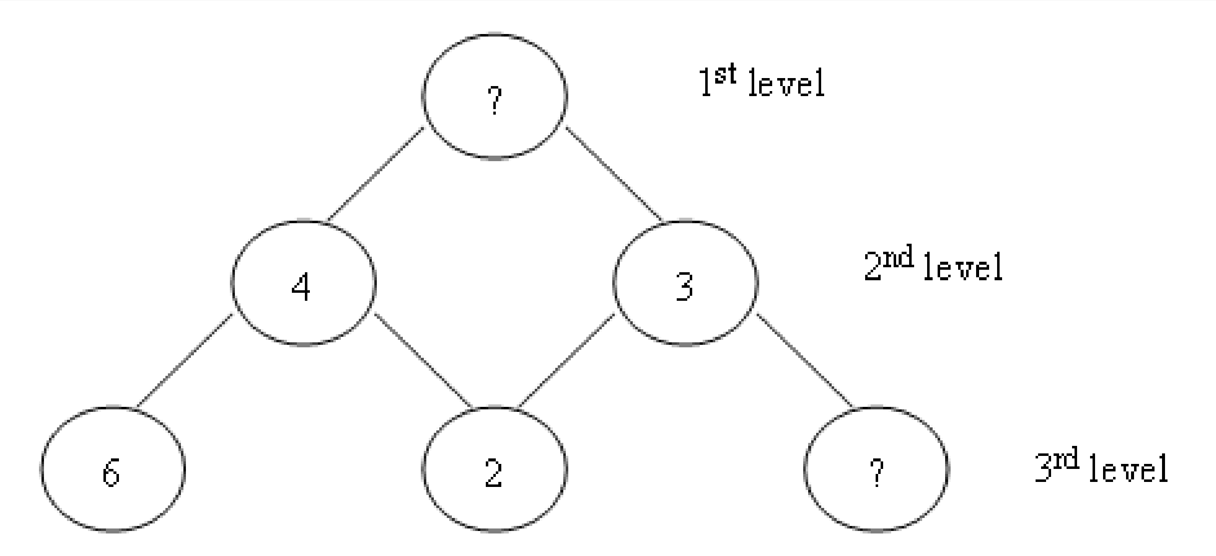
Example 5 – Toggle

- Try toggling all combinations of rows
- For each column, toggle it only if it has more black cells than white cells
- Check whether it has the most number of white cells

- Number of combinations = 2^N
- Time complexity : $O(N^2*2^N)$

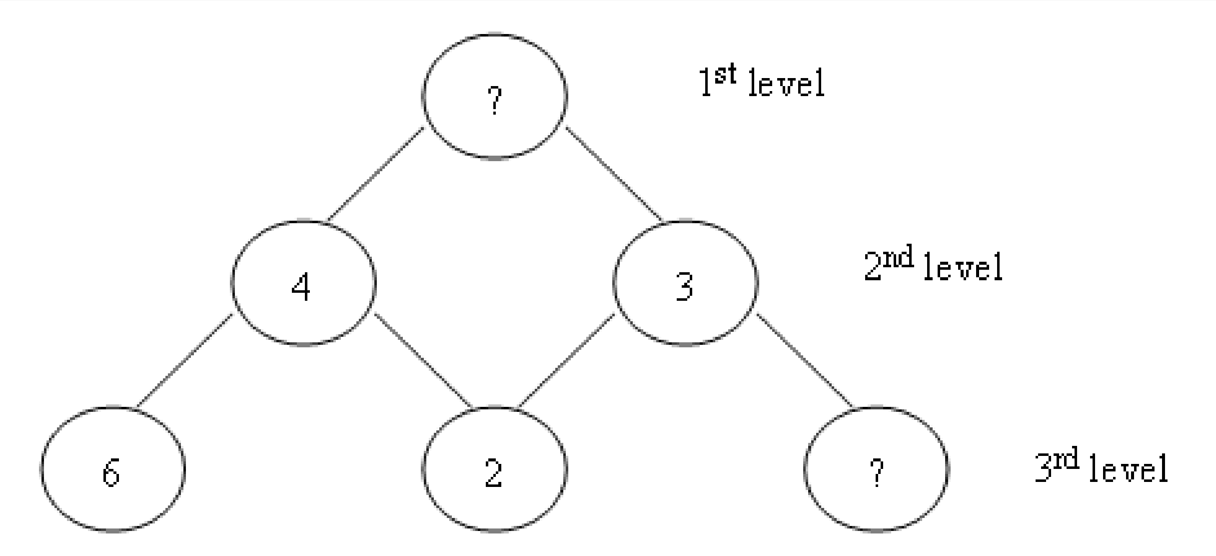
Example 6 – Magic Triangle

- There is a triangle with N levels (N <= 5)
- The ith level has i nodes
- Each node except those in the Nth level has two children



Example 6 – Magic Triangle

- Assign 1 to N*(N+1)/2 into the nodes
- The value of some nodes are predetermined
- Output an arrangement such that the value of each node = The absolute difference of the value of its 2 children



Example 6 – Magic Triangle

- Solution 1 :
- Try all permutations of numbers
- Check whether they fulfill the condition
- Number of permutations = $(N*(N+1)/2)!$
- Time complexity : $O((N*(N+1)/2)!)$

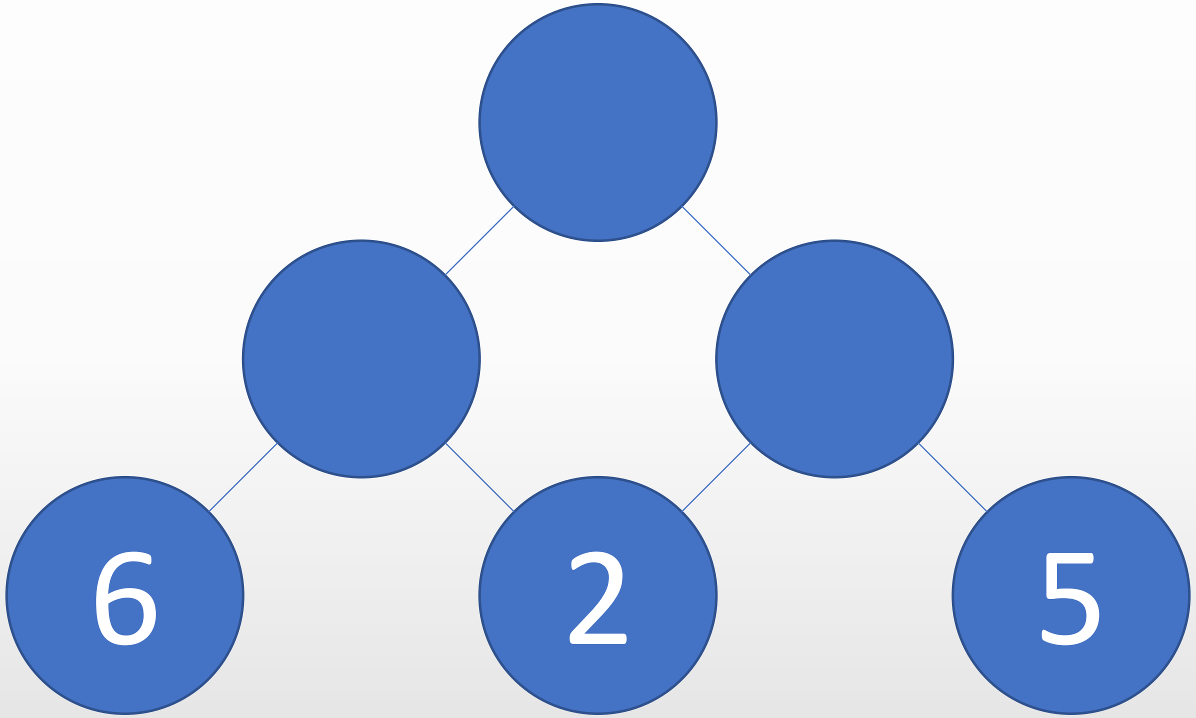
Example 6 – Magic Triangle

- When N = 5, number of permutations = 1307674368000
- Constant optimization is not enough
- Even by immediately discarding the permutations that does not match with the predetermined numbers, there are still too many
- A faster solution is needed

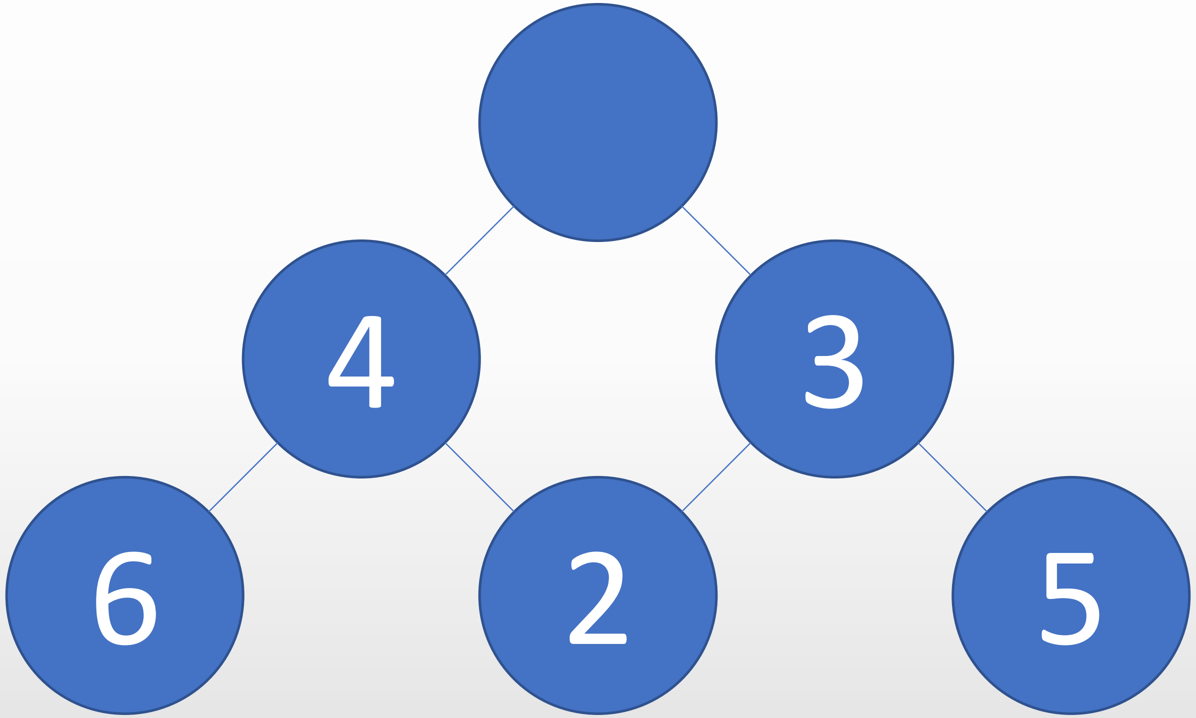
Example 6 – Magic Triangle

- Consider the values in the Nth level
- If these values are fixed, the values in the (N-1)th level can be calculated
- Use values in the (N-1)th level to calculate those in the (N-2)th level
- …
- …
- Use values in the 2nd level to calculate the value in the 1st level
- The whole arrangement is fixed!

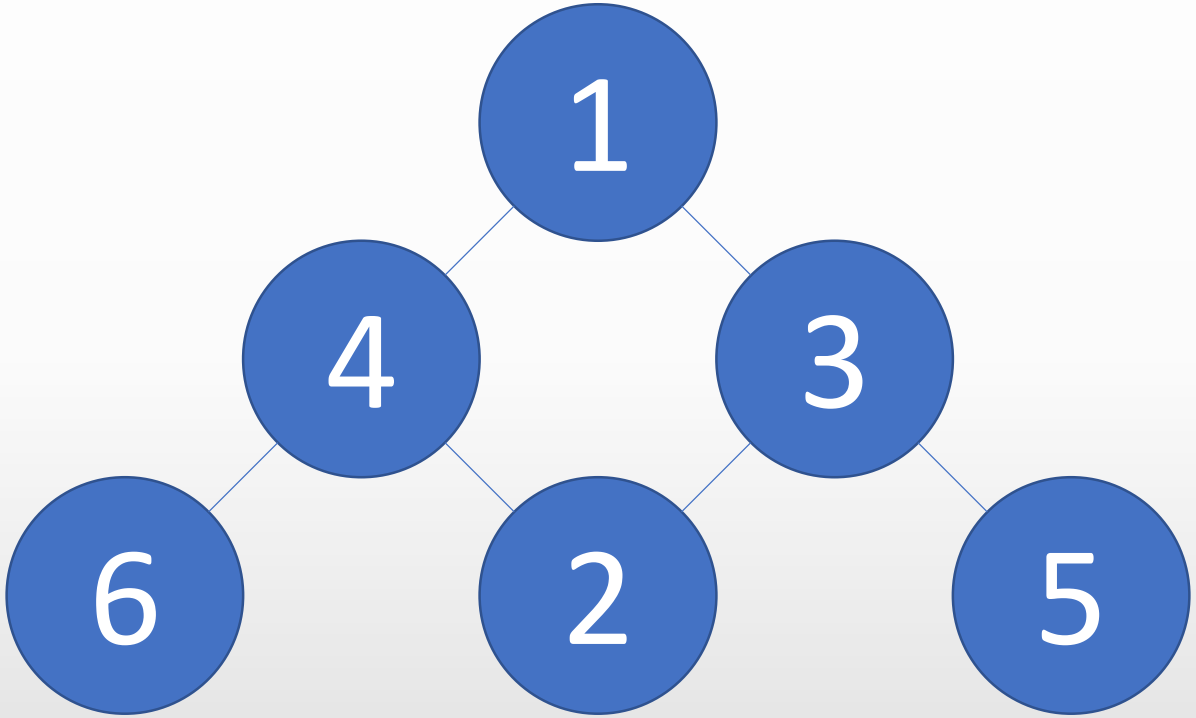
Example 6 – Magic Triangle



Example 6 – Magic Triangle



Example 6 – Magic Triangle



Example 6 – Magic Triangle

- Solution 2
- Exhaust all permutations of the Nth level
- Calculate the values of the whole triangle from the bottom to top
- Verify if the numbers are distinct

- Number of permutations = (N*(N+1)/2)PN
- When N is 5, the number of permutations = 360360
- Fast enough to solve the problem

Example

- HKOJ T161 Apple Race

Branch & Bound

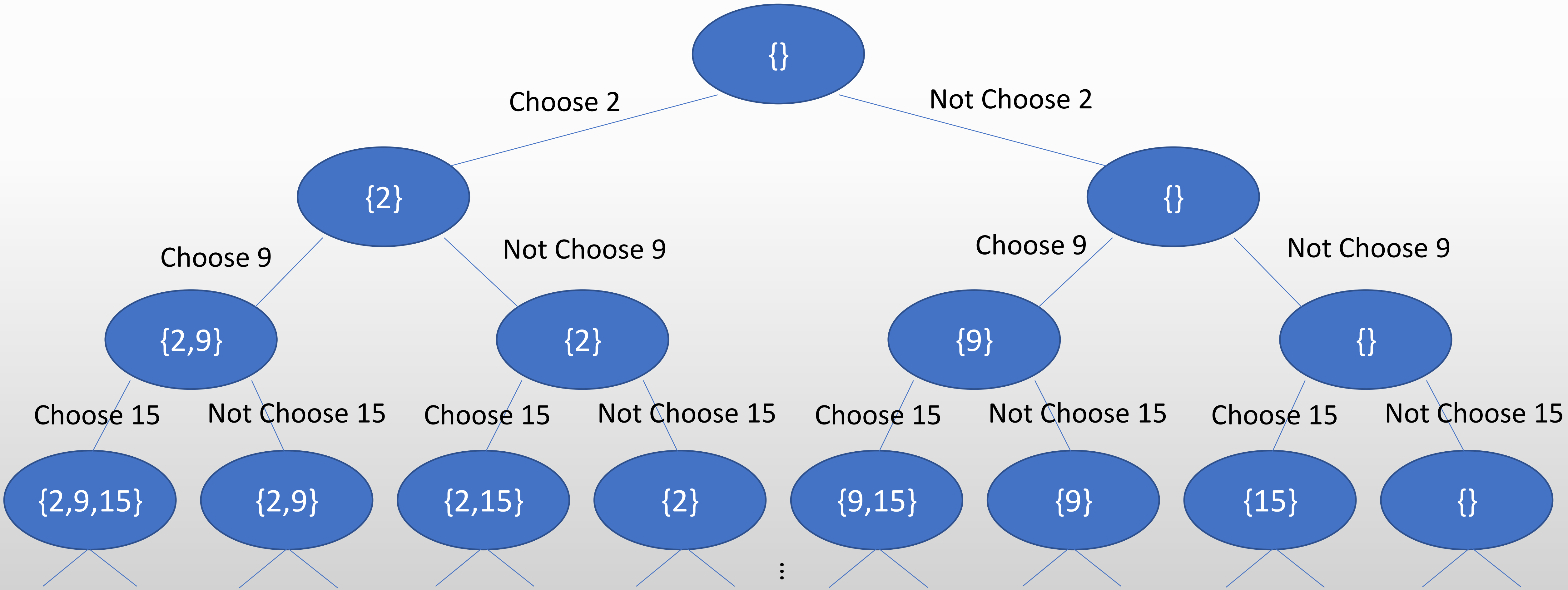
- During the searching, some invalid states may be visited
- When the intermediate state is known to be invalid, stop branching from it
- When the intermediate state gives an unsatisfied answer, stop branching from it

Example 2 – again

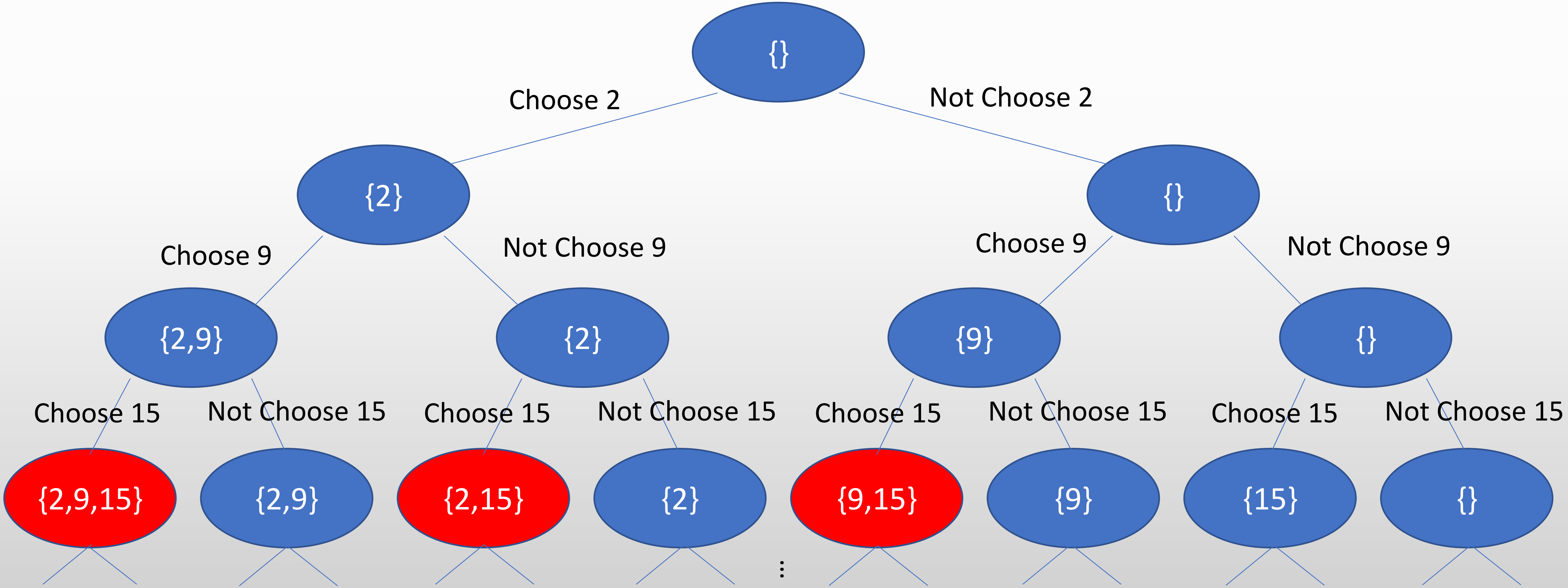
- Let’s go back to example 2
- Now A = {2, 9, 10, 3, 7, 2, 5, …} and M = 16
- Use the same technique again

Example 2 – again

- Some of the states are useless!

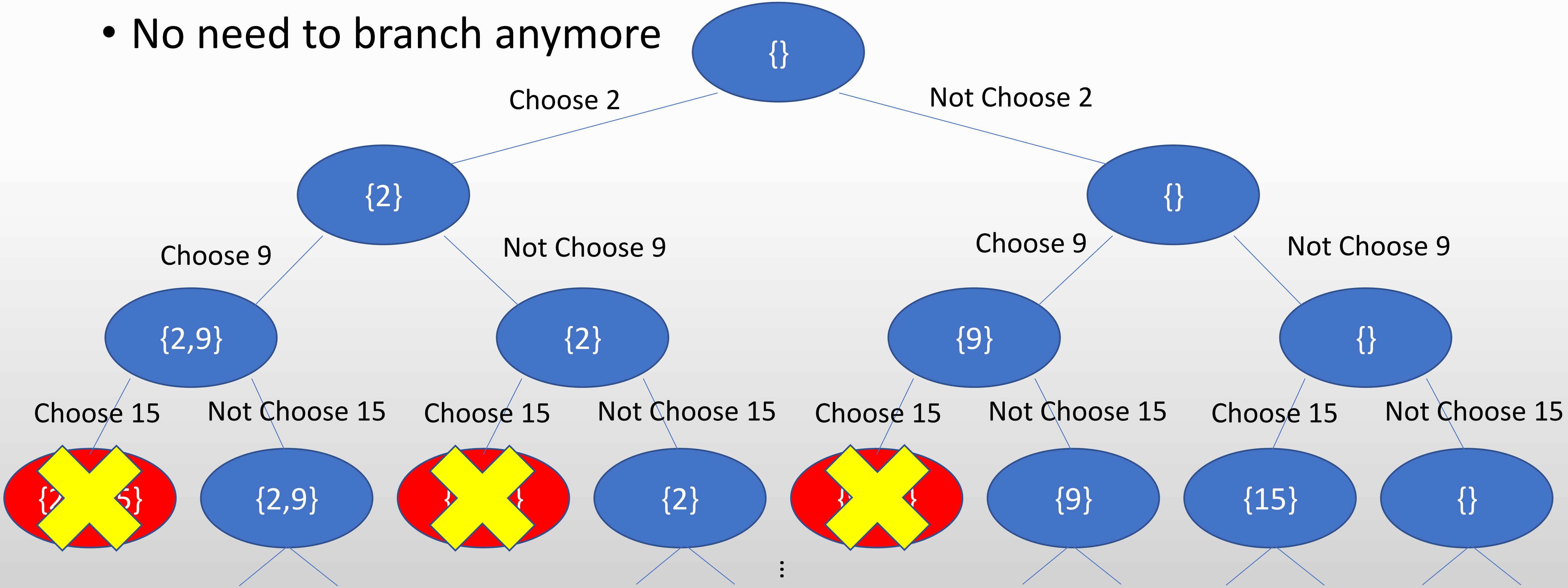


Example 2 – again



Example 2 – again

- Sum of the chosen numbers > 16
- No need to branch anymore



Example 2 – Pseudocode 2

```
Procedure exhaustion(int x, int sum)
    If sum > M then return
    If x <= N then
        choose[x] = true
        exhaustion(x + 1, sum + A[x])
        choose[x] = false
        exhaustion(x + 1, sum)
    Else
        if sum = M then
                output the numbers that are chosen
```

Example 2 – Pseudocode 3

```
Procedure exhaustion(int x, int sum)
    If sum > M then return
    If solution is found then return
    If x <= N then
        choose[x] = true
        exhaustion(x + 1, sum + A[x])
        choose[x] = false
        exhaustion(x + 1, sum)
    Else
        if sum = M then
                output the numbers that are chosen
```

Example 2 – again

- Constant optimization
- The time complexity is not affected much – still $O(2^N)$
- But much less operations are done
- Search meaningful states only

Example

- HKOJ 20750 8 Queens Chess Problem

Example 6 – Bin Packing

- You have N objects to be packed in some identical bins.
- Each of the bins has the same capacity C and can hold objects of total volume not exceeding C.
- Given the volume Vi of each object, at least how many bins are needed to pack all N objects?
- Example:
- N = 5, C = 50
- V = {10, 20, 30, 50, 20}
- The objects can be put into 3 bins : {10, 20}, {30, 20}, {50}

Example 6 – Bin Packing

- An obvious solution is to try all possibilities
- Put the first object into a new bin
- For the second object, try to put it into an existing and large enough bin, or try to put it into a new bin
- For the third object, try to put it into an existing and large enough bin, or try to put it into a new bin
- …
- Find the minimum number of bins among the possibilities

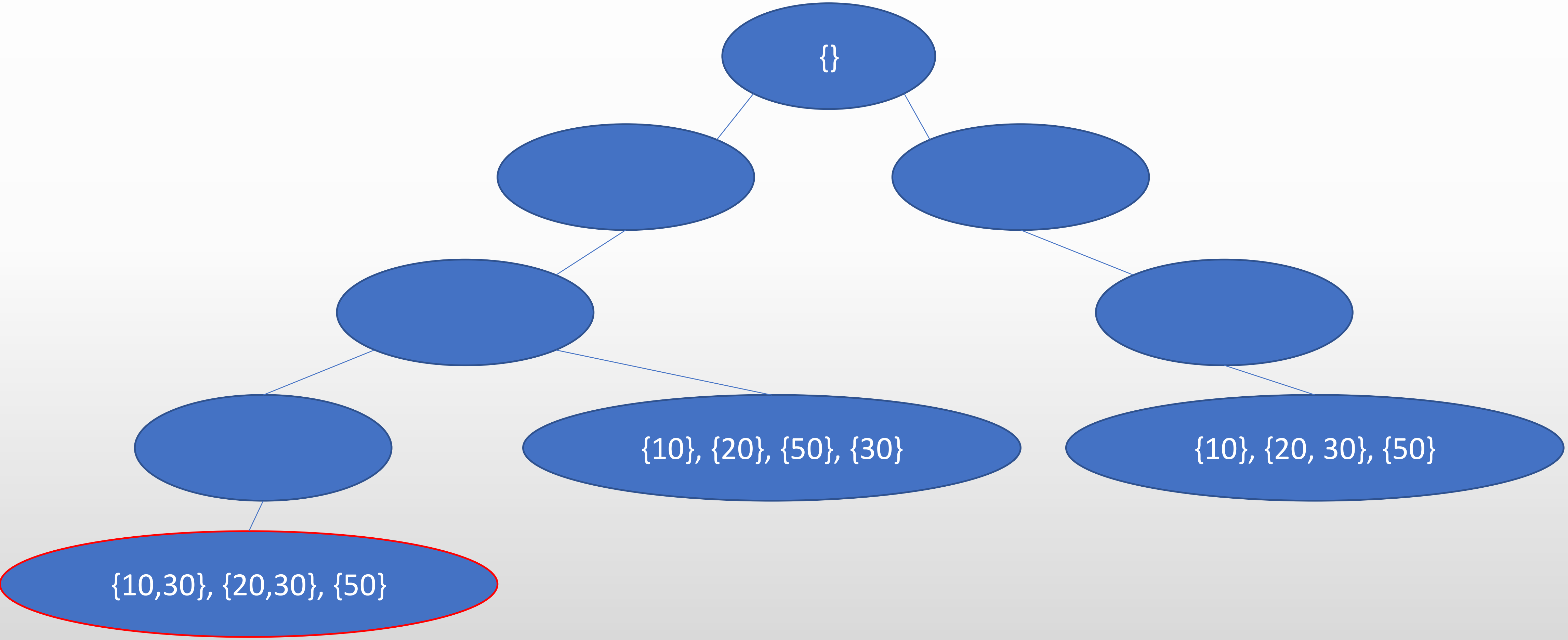
Example 6 – Bin Packing

- Not fast enough
- More optimization and pruning is needed

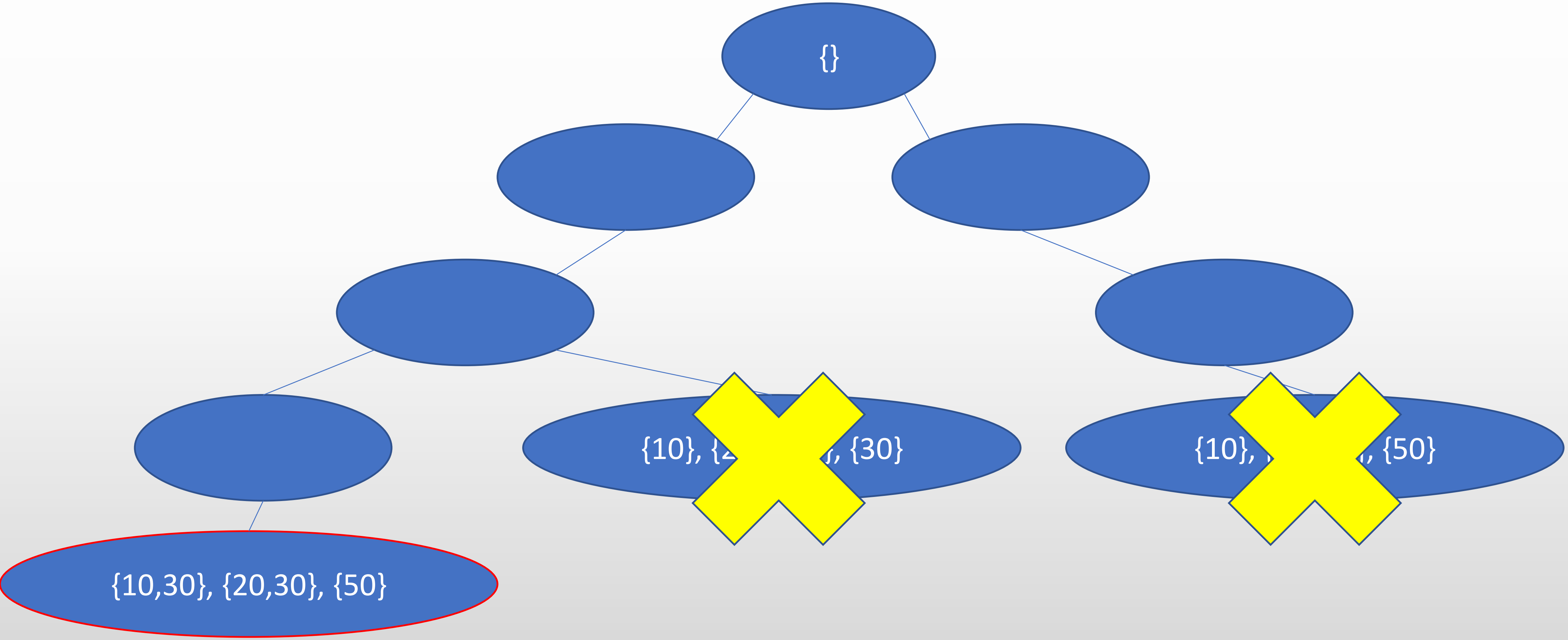
Example 6 – Bin Packing

- If the number of bins used in the current state is greater than or equal to the best answer by far, the state can be discarded
- The number of bins used will not decrease later
- It cannot help to reduce the answer

Example 6 – Bin Packing



Example 6 – Bin Packing

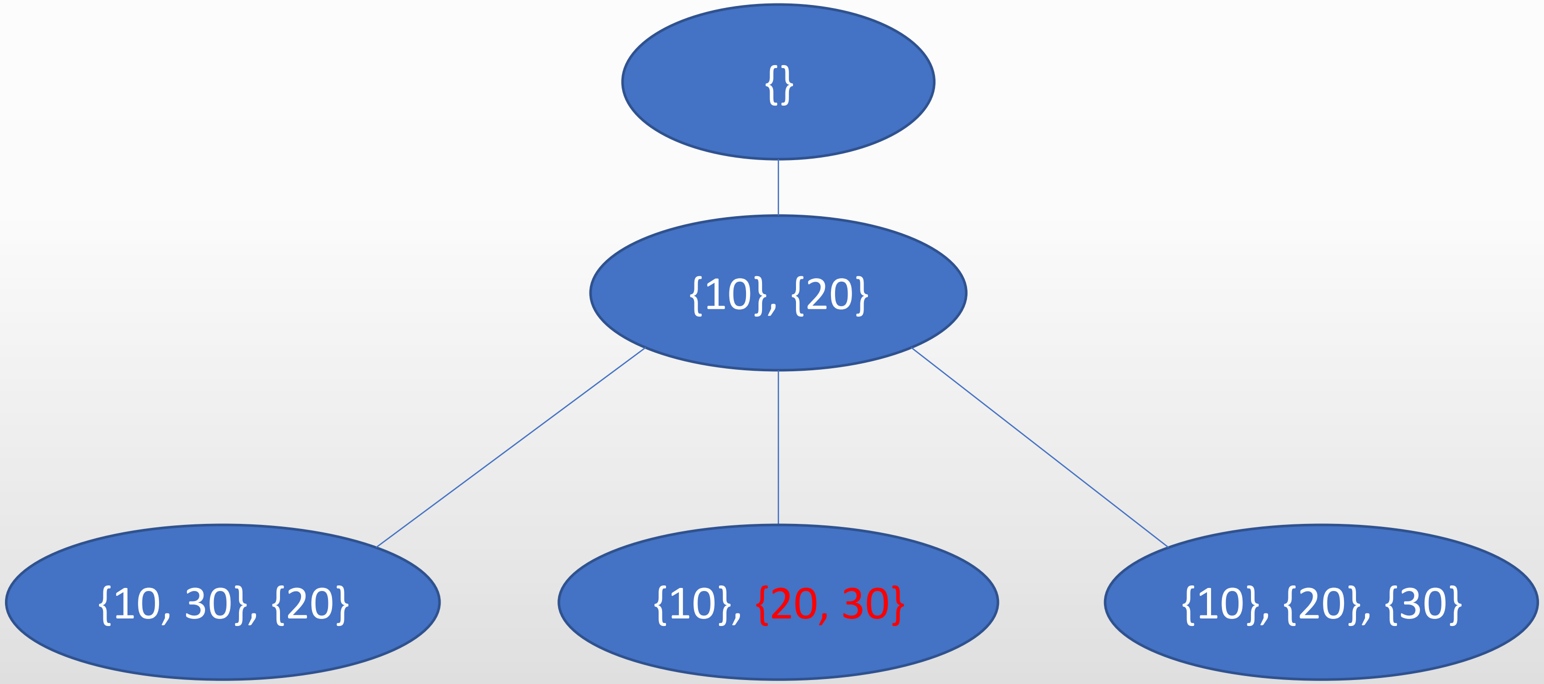


Example 6 – Bin Packing

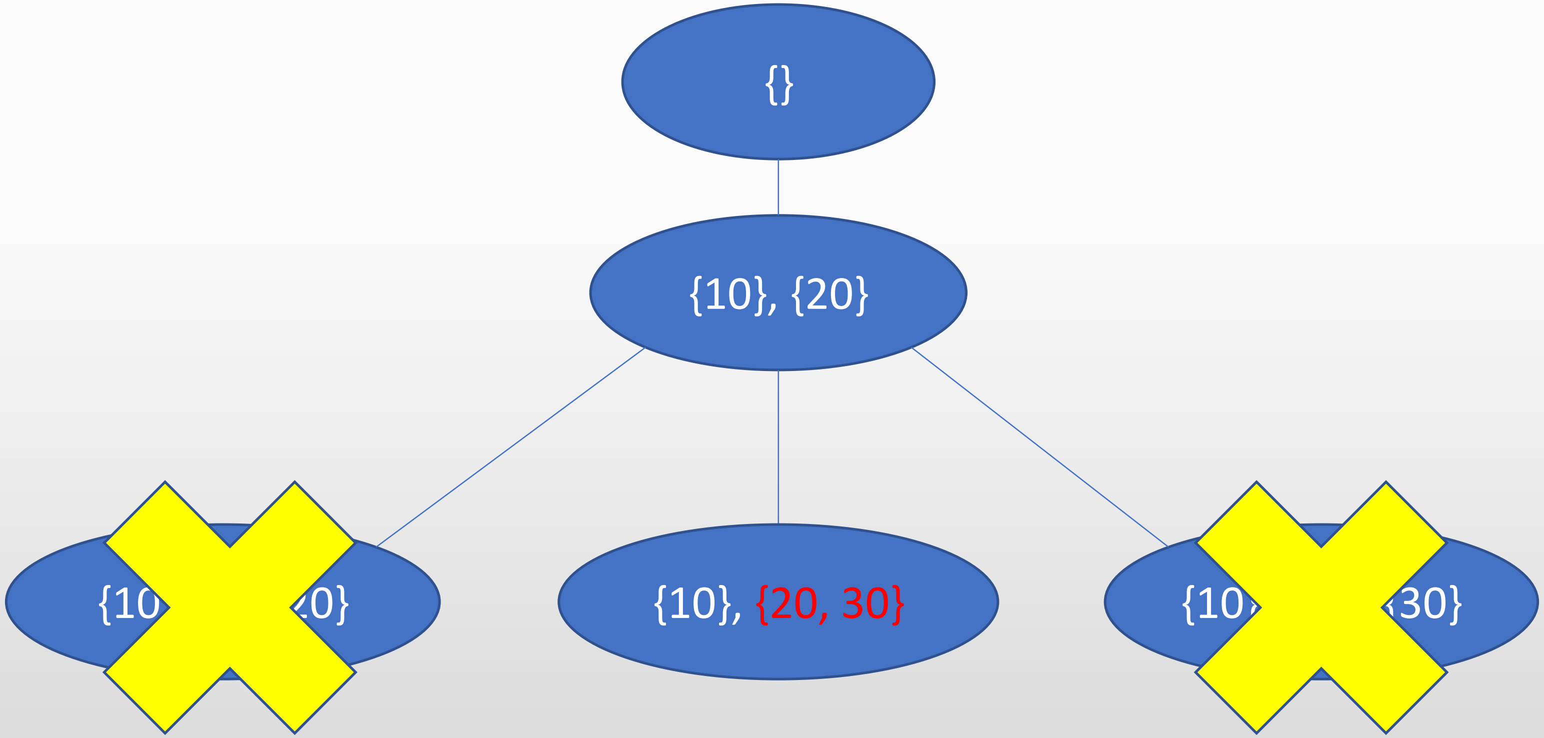
- Still not fast enough

- If the object can completely fill a bin, then the object must be put in that bin
- Use up the whole bin -> no capacity is wasted
- As it is the optimal way, we do not need to consider other cases

Example 6 – Bin Packing



Example 6 – Bin Packing



Example 6 – Bin Packing

- Still not fast enough

- Sort the objects in descending order of volume
- A suboptimal solution is generated faster
- No need to consider much about the distribution of small objects

{50}, {30}, {30}

{10} {20} {30}

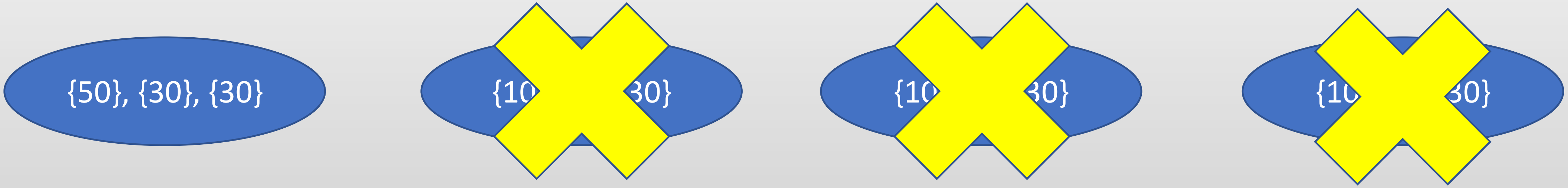
{10, 20}, {30}

{10}, {20, 30}

Example 6 – Bin Packing

- Still not fast enough

- Sort the objects in descending order of volume
- A suboptimal solution is generated faster
- No need to consider much about the distribution of small objects



Example 6 – Bin Packing

- Still not fast enough

- Maybe consider other implementations
- For each bin, try to choose some objects to put into it while keeping the volume not exceeding C
- Try to add one more bin if it is not enough
- Repeat until a valid arrangement is made

Example 6 – Bin Packing

- Many meaningless repetitions
- For example, {1, 2} {3, 4} and {3, 4} {1, 2} will be considered twice

- For a new bin, the first unselected object must be put into it
- Avoid permutation of bins

Example 6 – Bin Packing

- Still not fast enough?????

- Try to stop the program after a number of trials
- Do not guarantee a correct solution
- Usually give you a suboptimal answer

Example 6 – Bin Packing

- Using a mix of pruning and optimization, you may eventually solve the problem
- Be creative
- Consider different ways to cut the redundant states
- Luck may be important
- Random may be useful

Example

- HKOJ 01049 Chocolate

Practice problems

- HKOJ 01049 Chocolate
- HKOJ 01050 Bin Packing
- HKOJ 20750 8 Queens Chess Problem
- UVA 307 Sticks
- UVA 524 Prime Ring Problem

Summary

- Exhaustion is useful to solve subtasks with small constraints
- Easy to do even you have no idea about the full solution
- Give you some patterns about the answers

Summary

- Exhaustion is actually very common (only in subtasks)
- Appear in almost every HKOI-TFT
- Don’t hesitate to use it
- Don’t ignore it and spend all the time on other subtasks

Example

- HKOJ T171 Optimal Bowing