

R E P O R T

[SUBSET SUM]



학 과	컴퓨터공학부 컴퓨터공학전공
교수님	신인철 교수님
학 번	201911608
이 름	김지환
제출일	2023.06.04



목 차

1. Devide and Conquer 3
2. Dynamic Programming 7

Devide and Conquer

- 더 이상 나눌 수 없는 답으로 쪼개어 해답을 구함.

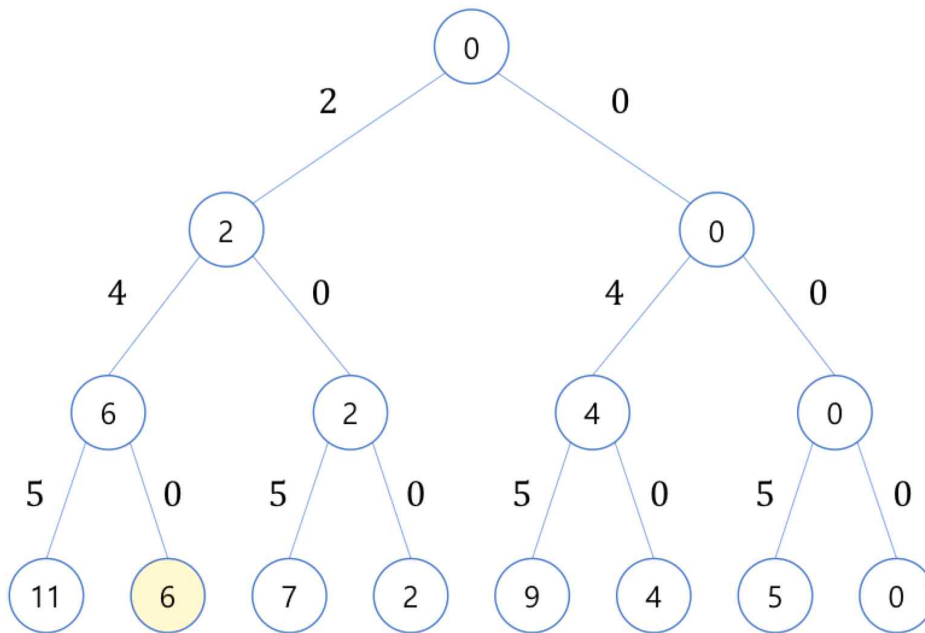
Subset 문제에서 더 이상 나눌 수 없는 답이란? - (a)

- 모든 부분집합을 찾을 때
- 수열의 마지막 원소를 탐색 했을 때

Subset 문제에서 답을 나누는 방법 - (b)

- {2, 4, 5}에서 모든 부분 집합을 구하기 위해서 idx, idx+1 번째를 탐색 해주면 된다.

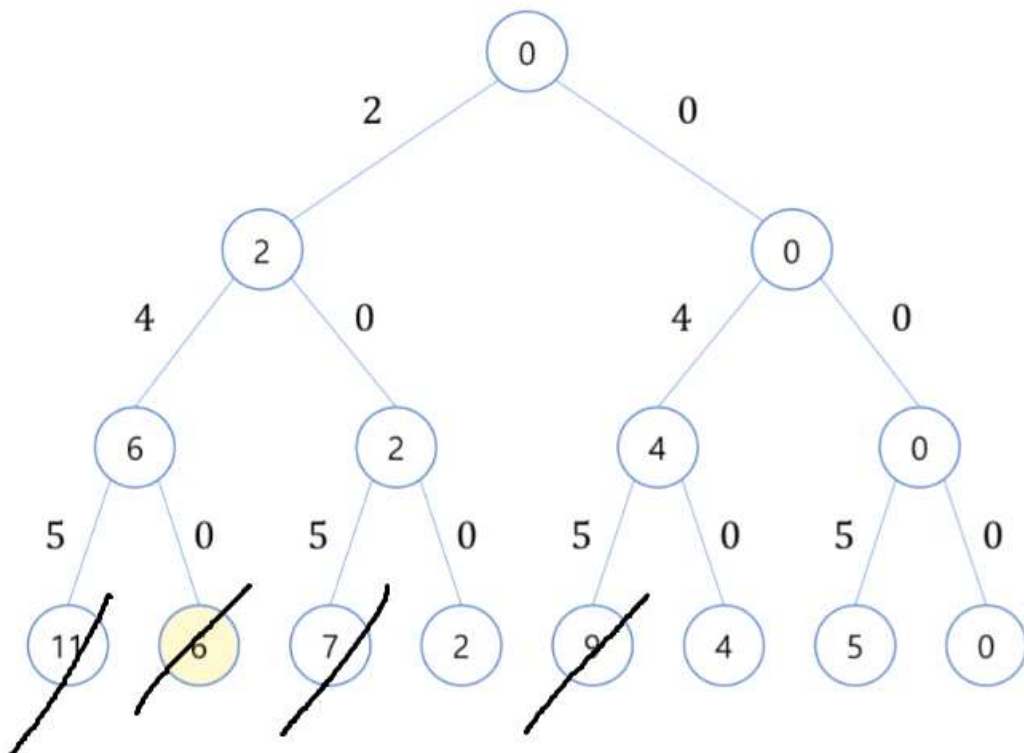
1. {2}, {4}
2. {2, 4}, {2}, {4, 5}, {4}
3. {2, 4, 5}, {2, 4}, {2}, {4, 5}, {5}, {4}



(a)에서 제시한 답을 이용하면 2^n 번 연산하게 된다.

Time Complexity 는 $O(2^n)$ 으로 예상할 수 있다.

개선 방안



내가 원하는 값(6)을 구함으로 이전 까지의 값 + 현재 값이 원하는 값을 초과할 경우 탐색하지 않은 메모제이션을 이용한 가지치기이다.
매우 개선되는 것은 아니지만 $O(2^n)$ 보다는 훨씬 빠른 시간을 갖는 방법이다.

Code 설명

```

#include <stdio.h>
#include <stdlib.h>

//SubSet을 찾기위한 Set 구성
typedef struct {
    int n;
    int size;
    int sum;
    int *arr;
    int *subset;
} Set;

//Set 초기화
void Init(Set *S) {
    S->arr = NULL;
    S->sum = 0;
  
```

```

        S->size = 0;
    }

    // 탐색 값, 수열 input
    void input(Set *S) {
        int n; char c;

        scanf("%d%c", &n, &c);
        S->n = n;

        // 마지막 문자가 개행일 경우 입력 종료.
        while(c!='\n') {
            scanf("%d%c", &n, &c);
            //동적으로 사이즈가 증가할 때마다 가변 할당
            S->arr = realloc(S->arr, sizeof(int)*(S->size+1));
            S->arr[S->size++] = n;

            S->sum+= n;

            if(n == 0) {
                int temp = S->arr[0];
                S->arr[0] = 0;
                S->arr[S->size-1] = temp;
            }
        }
        S->subset = calloc(S->size, sizeof(int));
    }

    //분할 작업 시작점
    void devide_and_conquer(Set *S, int sum, int n, int subsetSize, int total) {
        // 작업
    }

    int main() {
        Set S;

        Init(&S);
        input(&S);
    }

```

<pre> devide_and_conquer(&S, 0, 0, 0, S.sum); free(S.arr); free(S.subset); return 0; } </pre>
<p>1. 메모이제이션을 위한 subset 배열, set을 담은 arr 배열 등을 담은 구조체와 초기화.</p> <p>2. 동적으로 할당하며 배열을 입력받고 total 값을 갱신 후 subset 배열들을 0으로 초기화</p> <p>3. subset 작업</p> <pre> void devide_and_conquer(Set *S, int sum, int n, int subsetSize, int total) { if(sum == S->n) { int i; for(i=0; i<subsetSize; i++) { printf("%d ", S->subset[i]); } printf("\n"); return; } if(S->size == n total - sum - S->arr[n] > S->sum) return ; S->subset[subsetSize] = S->arr[n]; devide_and_conquer(S, sum + S->arr[n], n+1, subsetSize+1, total-S->arr[n]); S->subset[subsetSize] = 0; devide_and_conquer(S, sum, n+1, subsetSize, total); } </pre>
<p>위에 그림과 같은 가지치기 방식을 구현한 함수로 부분수열의 합이 찾고자하는 n과 같을 경우 subset들을 출력한다.</p> <p>그 후 기본적인 분할 작업으로 다시 나눈다.</p>

-> $O(2^n)$ 보다 조금 더 빠름.

실행결과

선택 D:\download\Devide_and_Conquer (201911608-김지환).exe

```
6 5 4 3 2 1
5 1
4 2
3 2 1
```

D:\download\Devide_and_Conquer (201911608-김지환).exe

```
6 6 5 4 3 2 1 0
0 5 1
0 4 2
0 3 2 1
0 6
5 1
4 2
3 2 1
6
```

D:\download\Devide_and_Conquer (201911608-김지환).exe

```
6 6 6 5 4 4 3 2 1
6
6
5 1
4 2
4 2
3 2 1
```

D:\download\Dynamic_Programming(201911608-김지환).exe

```
6 6 6 5 4 4 3 2 1
6
6
2 4
2 4
1 5
1 2 3
```

Dynamic Programming

- 동적 프로그래밍은 겹치는 부분문제나 최적 부분 구조를 가질 경우 해결 할 수 있다.

겹치는 부분 문제?

- Divide and Conquer에서 (b)를 확인하면 겹치는 부분 문제임을 알 수 있다.
- 더하는 값들이 중복된다.

최적 부분 구조?

- $2 + 4 + 5 = 11$ 이고
- $2 + 5 + 4 = 11$ 이다.
- 11일 구할 수 있는 방법은 여러개지만 그 중 하나만 구하면 되므로 최적 부분 구조이다.

DP 점화식

if ($A[i-1] > j$)

$dp[i][j] = dp[i-1][j]$

else

$dp[i][j] = dp[i-1][j] \text{ OR } dp[i-1][j-\text{set}[i-1]]$

set = {3, 4, 5, 2} 일 때,

1. set[idx] 는 무조건 set[idx] + 0으로 만들 수 있으므로 set[idx]와 set[0] = true이다.
2. set[idx]보다 작을 경우 set[idx]-작은값으로 되는 케이스인지 결과 값을 가져온다.
3. set[idx]보다 클 경우 큰값-set[idx]로 되는 케이스인지 결과 값을 가져온다.

	0	1	2	3	4	5	6
no element (0)	T	F	F	F	F	F	F
0 (3)	T	F	F	T	F	F	F
1 (4)	T	F	F	T	T	F	F
2 (5)	T	F	F	T	T	T	F
3 (2)	T	F	T	T	T	T	T

Code 설명

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// SubSet을 찾기 위한 Set 구성
typedef struct {
    int n;
    int size;
    int *arr;
    int *subset;
    bool **dp;
} Set;

// Set 초기화
void Init(Set *S) {
    S->arr = NULL;
    S->size = 0;
}

// 탐색 값, 수열 input
void input(Set *S) {
    int n, i;
    char c;

    scanf("%d%c", &n, &c);
    S->n = n;

    // 마지막 문자가 개행일 경우 입력 종료.
    while(c!='\n') {
        scanf("%d%c", &n, &c);
        // 동적으로 사이즈가 증가할 때마다 가변 할당
        S->arr = realloc(S->arr, sizeof(int)*(S->size+1));
        S->arr[S->size++] = n;

        if(n == 0) {
            int temp = S->arr[0];
            S->arr[0] = 0;
            S->arr[S->size-1] = temp;
        }
    }
}
```

```

        S->subset = calloc(S->n, sizeof(int));
        S->dp = (bool **)calloc((S->size + 1), sizeof(bool *));

        for (i = 0; i <= S->size; i++) {
            S->dp[i] = (bool *)calloc((S->n + 1), sizeof(bool));
        }
    }

    //백 트래킹
    //DP 테이블 생성하던 반대로
    void findSubsets(Set *S, int subsetSize, int i, int j) {
        //
    }

    // DP테이블 생성
    void subsetDP(Set *S) {
        //
    }

    int main() {
        Set S;
        int i, j;

        Init(&S);
        input(&S);

        subsetDP(&S);
        free(S.arr);

        for(i = 0; i <= S.size; i++) free(S.dp[i]);
        free(S.dp);

        return 0;
    }

```

기존 분할정복 코드에서 추가적으로 DP 테이블을 생성하는 명령과 DP 테이블 갱신 작업하는 함수와 갱신 된 DP 테이블을 토대로 백트래킹 하여 Subset 결과를 찾는 함수로 구성했다.

```
void subsetDP(Set *S) {
```

```

int i, j;

for (i = 0; i <= S->size; i++) S->dp[i][0] = true;

for (i = 1; i <= S->size; i++) {
    int target = S->arr[i-1];
    for (j = 1; j <= S->n; j++) {
        if (j < target) {
            S->dp[i][j] = S->dp[i - 1][j];
        }
        else {
            S->dp[i][j] = S->dp[i - 1][j] || S->dp[i - 1][j - target];
        }
    }
}

// 백트래킹
findSubsets(S, 0, S->size, S->n);
}

```

1. 찾는 값이 0일 때는 무조건 True임을 알 수 있다.
 $0 + n = n$
2. 0을 제외한 1번째 인덱스부터 점화식을 그대로 작성한다.
3. 점화식을 도출한 과정을 반대로 백트래킹 하여 subset을 찾는다.

```

void findSubsets(Set *S, int subsetSize, int i, int j) {
    int target = S->arr[i-1];
    if (i == 0 && j == 0) {
        for(i; i<subsetSize; i++) printf("%d ", S->subset[i]);
        printf("\n");
        return;
    }

    if (i > 0 && S->dp[i - 1][j]) {
        findSubsets(S, subsetSize, i - 1, j);
    }

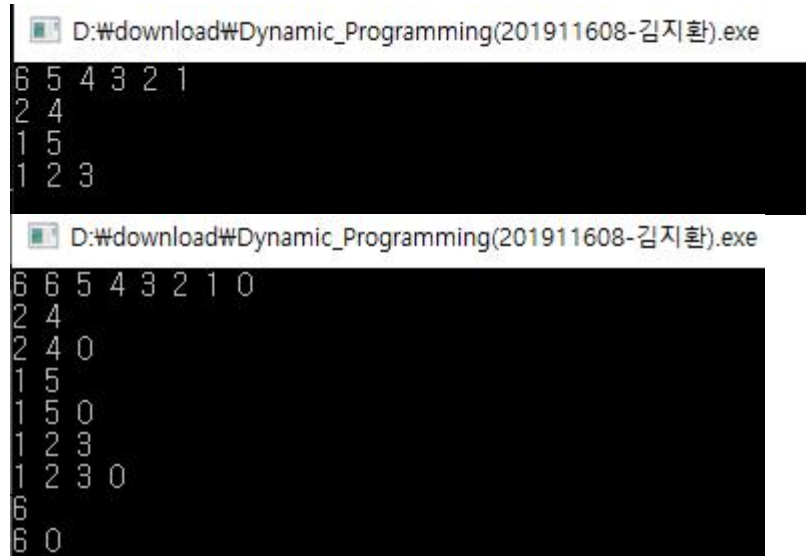
    if (i > 0 && j >= target && S->dp[i - 1][j - target]) {
        S->subset[subsetSize] = target;
        findSubsets(S, subsetSize+1, i - 1, j - target);
    }
}

```

```
S->subset[subsetSize] = 0;  
}  
}
```

-> $O(\text{배열사이즈} * \text{findNum})$

실행결과



D:\download\Dynamic_Programming(201911608-김지환).exe

```
6 5 4 3 2 1  
2 4  
1 5  
1 2 3
```

D:\download\Dynamic_Programming(201911608-김지환).exe

```
6 6 5 4 3 2 1 0  
2 4  
2 4 0  
1 5  
1 5 0  
1 2 3  
1 2 3 0  
6  
6 0
```