

#2. 그리디 기법 2

나정휘 (jhnah917)

<https://justicehui.github.io/>

Exchange Argument

Exchange Argument

- BOJ 11399 ATM의 증명 방법을 일반화한 것이라고 생각할 수 있음
- 특정 비교 기준으로 봤을 때 inversion이 없으면 최적임을 증명할 수 있다면
- 인접한 두 원소의 순서를 결정하는 것으로 전체 원소의 순서를 결정할 수 있음
 - ex. 버블 정렬, 삽입 정렬, ...
- 버블 정렬과 다른 일반적인 비교 기반 정렬의 결과물을 동일하므로
 - ex. quick sort, heap sort, merge sort, ...
- 비교 함수를 잘 작성한 뒤 `std::sort`를 사용하면 됨

그리디 예시 - 5

BOJ 14908 구두 수선공

- i 번째 작업을 수행하는데 T_i 일 걸림
- i 번째 작업의 완료가 하루 지연될 때마다 S_i 원 벌금 내야 함
- 벌금을 최소로 하는 작업 순서를 정하는 문제

그리디 예시 - 5

BOJ 14908 구두 수선품

- $\text{sum}(T_{1..k-1}) * S_k + (\text{sum}(T_{1..k-1}) + T_k) * S_{k+1}$
- $\text{sum}(T_{1..k-1}) * S_{k+1} + (\text{sum}(T_{1..k-1}) + T_{k+1}) * S_k$

$T_{1..k-1}$	T_k	T_{k+1}	$T_{k+1..n}$
$T_{1..k-1}$	T_{k+1}	T_k	$T_{k+1..n}$

- k번째 원소가 k+1번째보다 먼저 오기 위해서는 $T_k * S_{k+1} \leq T_{k+1} * S_k$ 가 되어야 함
- $T_k / S_k \leq T_{k+1} / S_{k+1}$ 이 되도록 정렬해야 함
- T_i / S_i 오름차순 정렬

그리디 예시 - 6

BOJ 2180 소방서의 고민

- 일차함수 $f_i(x) = a_i x + b_i$ 가 여러 개 주어짐 ($a_i, b_i > 0$)
- $x = 0$ 에서 시작해서 $x \leftarrow x + f_i(x)$ 를 한 번씩 적용할 때
- 최종 결과의 최솟값을 구하는 문제

그리디 예시 - 6

BOJ 2180 소방서의 고민

- $(a_{k+1}+1)((a_k+1)x + b_k) + b_{k+1}$
- $(a_k+1)((a_{k+1}+1)x + b_{k+1}) + b_k$

x	$a_kx + b_k$	$a_{k+1}x + b_{k+1}$	f
x	$a_{k+1}x + b_{k+1}$	$a_kx + b_k$	f

- $a_{k+1}x + a_k a_{k+1}x + a_{k+1}b_k + x + a_kx + b_k + b_{k+1}$
- $a_kx + a_k a_{k+1}x + a_k b_{k+1} + x + a_{k+1}x + b_{k+1} + b_k$
- $a_{k+1}b_k \leq a_k b_{k+1}$ 이 되도록 정렬
- $b_k / a_k \leq b_{k+1} / a_{k+1}$ 이 되도록 정렬
- b_i / a_i 오름차순 정렬

우선순위 큐

우선순위 큐 (Priority Queue)

- Queue: 먼저 들어온 데이터가 먼저 나가는 자료구조
- Priority Queue: 우선순위가 높은 데이터가 먼저 나가는 자료구조
 - 원소 삽입
 - 가장 큰 원소 검색
 - 가장 큰 원소 제거
- 목표: N개의 데이터가 있을 때 세 연산을 모두 $O(\log N)$ 정도에 처리하는 것

우선순위 큐

힙 구조

- 각 정점의 값이 자식 정점보다 큰 트리 구조
- 주로 사용하는 건 이진 힙 (Binary Heap)
 - 완전 이진 트리(Complete Binary Tree) 형태
 - 루트 : 1번 인덱스
 - x 의 부모 : $x / 2$
 - x 의 왼쪽/오른쪽 자식 : $2x, 2x+1$
- 이진 힙 외에도 다양한 힙 자료구조가 있음
 - binomial heap, leftist heap, pairing heap, fibonacci heap, thin heap, ...

우선순위 큐

std::priority_queue

- #include <queue>
- 이진 힙으로 구현되어 있음
- 기본값은 max heap - 가장 큰 원소가 맨 위에 있음
 - min heap: priority_queue<int, vector<int>, greater<>> pq;
 - max heap: priority_queue<int, vector<int>, less<>> pq;
- pq.push(x)
- pq.pop(), pq.top()
- pq.size(), pq.empty()

그리디 예시 - 7

허프만 코드

- N가지의 문자로 구성된 텍스트 파일을 압축해야 함
- i번째 문자의 등장 횟수는 $F[i]$ 이고, 각 문자를 적당한 이진수로 표현해서 파일의 용량을 최소화하려고 함
- 즉, 심볼들의 출현 빈도가 정해져 있을 때, 심볼 단위로 인코딩해서 전체 길이를 최소화하는 문제
- prefix code: 어떠한 코드도 다른 코드의 prefix가 되지 않는 코드
 - 0, 10, 1100, 1101, 11100 은 prefix code
 - 0, 10, 110, 1100 은 prefix code가 아님
- prefix code의 장점: 앞에서부터 그리디하게 매칭하는 방식으로 디코딩할 수 있음
 - 앞에서부터 한 글자씩 읽으면서, 처음으로 완성되는 코드를 가져가는 방식으로 디코딩 가능
 - ex. 등장할 수 있는 코드가 0, 10, 1100, 1101, 11100
 - 인코딩된 문자열이 0110010110111100010 이라면?
 - 0 / 1100 / 10 / 1101 / 11100 / 0 / 10 으로 나눠서 디코딩하면 됨

그리디 예시 - 7

허프만 코드

- prefix code tree

- 0, 10, 1100, 1101, 11100

ACBDEAB

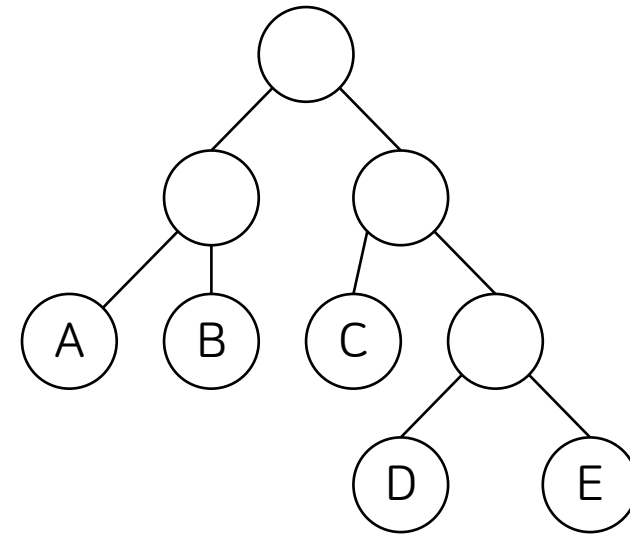
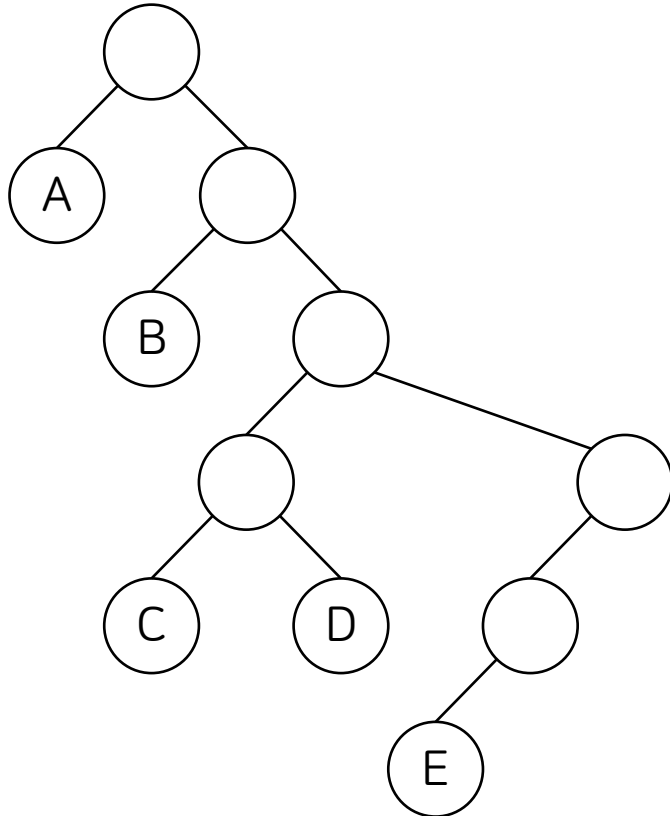
0 1100 10 1101 11100 0 10

- 00, 01, 10, 110, 111

ACBDEAB

00 10 01 110 111 00 01

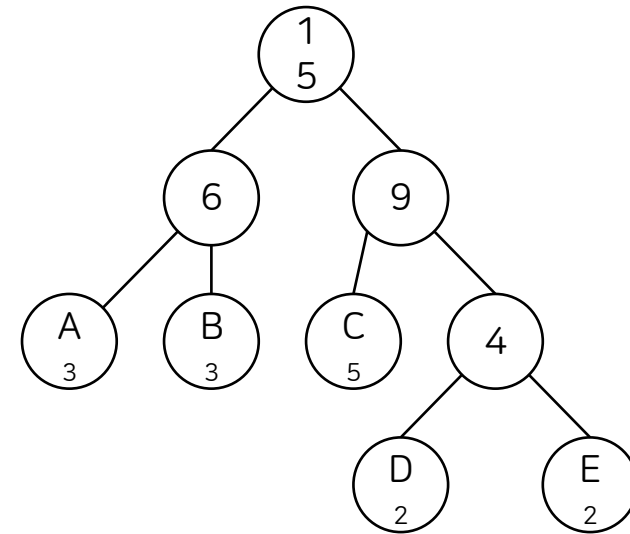
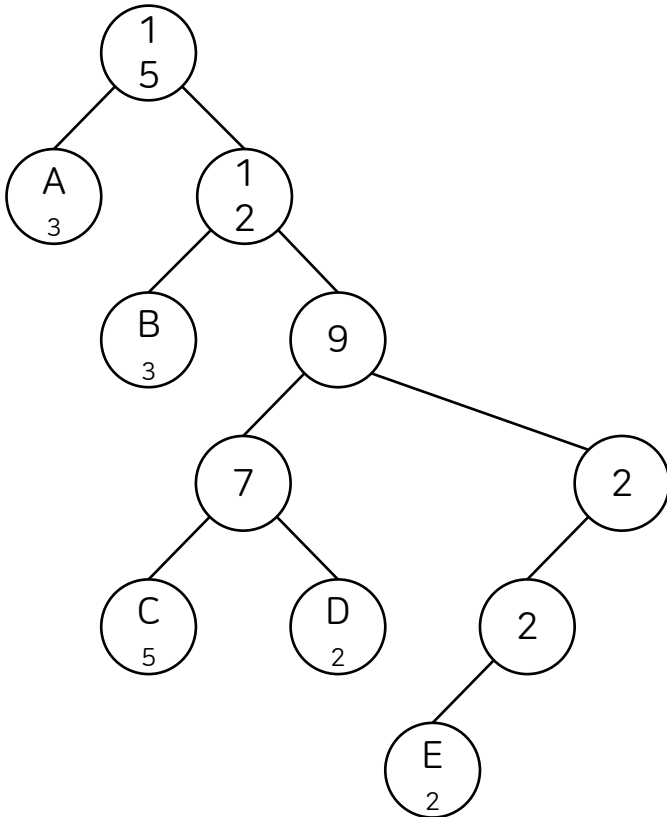
- 가중치 합이 최소가 되도록 트리를 구성하는 문제



그리디 예시 - 7

허프만 코드

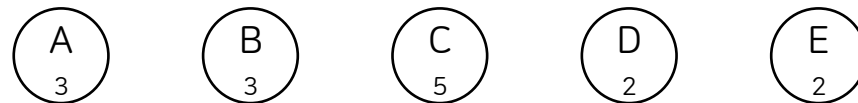
- 가중치 합이 최소가 되도록 트리를 구성하는 문제
 - 가중치: 각 정점마다, 그 정점을 루트로 하는 서브 트리 아래에 있는 심볼의 등장 횟수의 합
 - $F[A] = 3, F[B] = 3, F[C] = 5, F[D] = 2, F[E] = 2$ 이면?



그리디 예시 - 7

허프만 코드

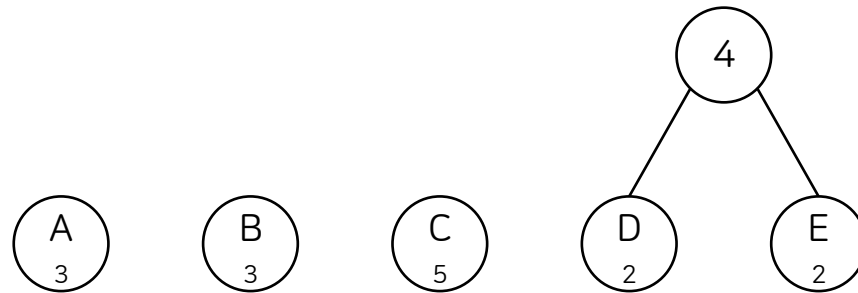
- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

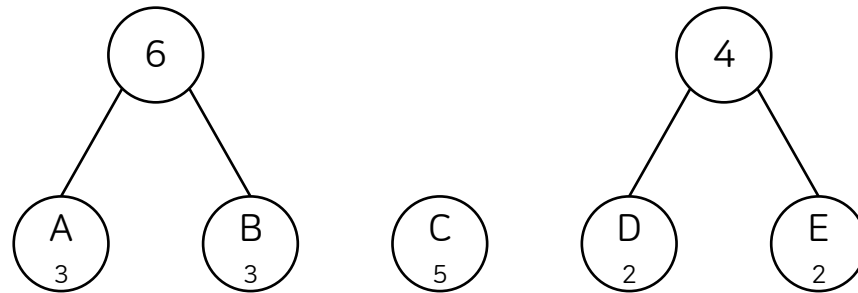
- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

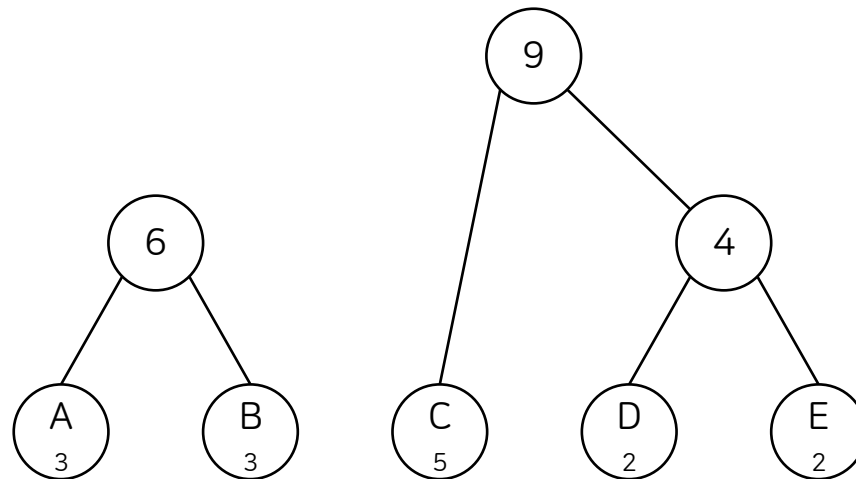
- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

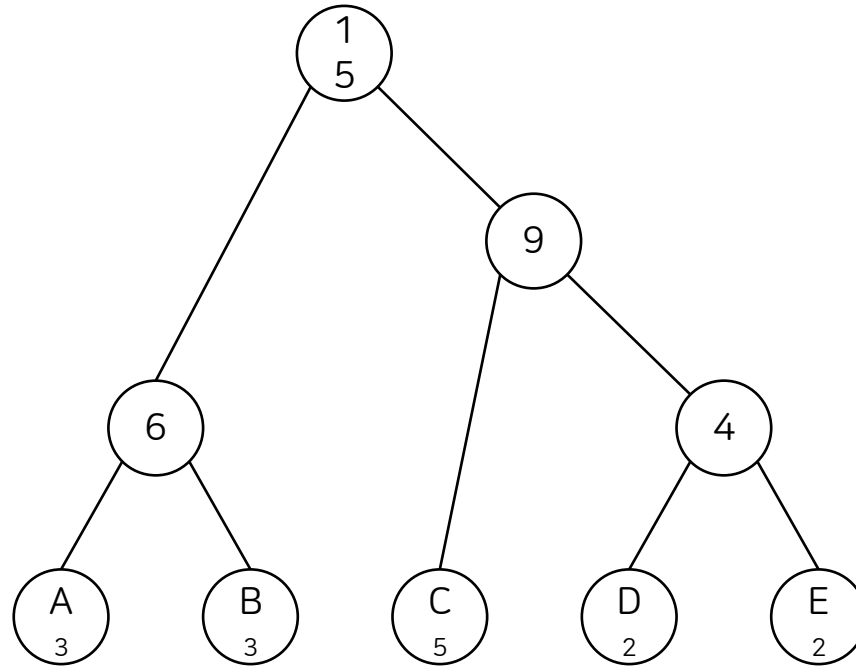
- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복



그리디 예시 - 7

허프만 코드

- 그리디 알고리즘
 - 처음에는 각 심볼마다 자신만 있는 트리로 초기화
 - 아직 트리가 2개 이상 있으면, 가중치가 가장 작은 두 루트 정점을 합쳐서 새로운 트리를 만드는 것을 반복

```
void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    int res = 0;
    priority_queue<ll, vector<ll>, greater<>> pq;
    for(int i=1; i<=N; i++) pq.push(A[i]);
    while(pq.size() > 1){
        int u = pq.top(); pq.pop();
        int v = pq.top(); pq.pop();
        res += u + v;
        pq.push(u + v);
    }
    cout << res << "\n";
}
```

그리디 예시 - 8

예시 8

- 양의 정수로 구성된 배열 $A[1..N]$ 이 주어짐
- 원소를 K 개 선택할 때, 선택한 원소들의 합을 최대화하는 문제
- 정렬을 이용해 간단하게 해결할 수 있지만, 그리디 기법으로 해결해 보자
- ex. $A = [3, 2, 4, 6, 1, 5], K = 3$
 - 앞에 있는 원소부터 차례대로 처리
 - $i = 1$: 3을 넣으면 합이 증가하므로 정답에 추가 현재 선택한 수: {3}
 - $i = 2$: 2를 넣으면 합이 증가하므로 정답에 추가 현재 선택한 수: {3, 2}
 - $i = 3$: 4를 넣으면 합이 증가하므로 정답에 추가 현재 선택한 수: {3, 2, 4}
 - $i = 4$: 6을 넣고 싶지만 이미 3개의 원소를 선택한 상황
 - 앞에서 한 행동을 하나 무르면 6을 넣을 수 있음
 - 가장 작은 수를 넣은 행동을 취소하자 현재 선택한 수: {3, 6, 4}
 - $i = 5$: 1은 선택된 가장 작은 수보다 작으므로 넘어감
 - $i = 6$: 선택된 가장 작은 수인 3을 없애고 5 추가 현재 선택한 수: {5, 6, 4}
- 작은 값부터 제거하는 우선 순위 큐를 이용해 $O(N \log N)$ 에 해결 가능

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N 개 있음
- i 번째 작업은 $D[i]$ 초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 $V[i]$ 원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제
- 틀린 전략
 - 마감 기한 오름차순으로 정렬한 다음, 지금까지 수행한 작업이 $D[i]$ 개 미만이라면 i 번째 작업을 수행
 - $D = \{1, 2, 2\}$, $V = \{1, 3, 2\}$ 이면 이 전략은 1/2번 작업을 선택하지만, 2/3번 작업을 선택하는 것이 최적
 - 이 전략은 작업의 개수를 최대화하지만 가중치를 최대화하지 못함

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업은 $D[i]$ 초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 $V[i]$ 원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제
- 틀린 전략
 - 마감 기한 오름차순으로 정렬한 다음, 지금까지 수행한 작업이 $D[i]$ 개 미만이라면 i번째 작업을 수행
 - $D = \{1, 2, 2\}$, $V = \{1, 3, 2\}$ 이면 이 전략은 1/2번 작업을 선택하지만, 2/3번 작업을 선택하는 것이 최적
 - 이 전략은 작업의 개수를 최대화하지만 가중치를 최대화하지 못함
- 관찰
 - $D[a] \leq D[b]$ 이면 일단 a번 작업을 수행하기로 결정한 다음
 - 혹시라도 a 대신 b 를 수행하는 것이 이득이었다면, a를 수행하려고 했던 시간에 b를 수행해도 됨
 - 예시 8처럼 가중치가 작은 작업을 버리는 전략?

그리디 예시 - 9

예시 9

- 수행하는데 1초가 걸리는 작업이 N개 있음
- i번째 작업은 $D[i]$ 초 안에 끝내야 하며, 마감 기한 안에 작업을 완료하면 $V[i]$ 원을 받을 수 있음
- 얻을 수 있는 금액을 최대화하는 문제
- 관찰
 - $D[a] \leq D[b]$ 이면 일단 a번 작업을 수행하기로 결정한 다음
 - 혹시라도 a 대신 b를 수행하는 것이 이득이었다면, a를 수행하려고 했던 시간에 b를 수행해도 됨
 - 예시 8처럼 가중치가 작은 작업을 버리는 전략?
- 올바른 전략
 - 마감 기한 오름차순으로 정렬
 - 지금까지 수행한 작업이 $D[i]$ 개 미만이면 i번째 작업을 수행
 - 지금까지 수행한 작업이 $D[i]$ 개면 가장 가중치가 작은 작업을 취소하고 i번째 작업을 수행

그리디 예시 - 10

예시 10

- N개의 작업이 있음
- i번째 작업을 수행하는데 $T[i]$ 초가 걸리며, $D[i]$ 초 안에 끝내야 함
- 수행하는 작업의 개수를 최대화하는 문제

- 풀이는 직접 고민해 보자.