자료구조 별 속도 실험

2022년 8월 28일 일요일 오전 1:05

◘ STL List / Vector / 정적할당 list

9개 TC, 100~1M 개의 숫자를 넣고 컨테이너 초기화
List : 70ms / Vec : 12ms / 정적할당 : 3ms

결론 : 삭제가 빈번하지 않다면 STL vec나 정적할당!

- **중간에 지워야 하는게 아니라면 STL List는 지양하라!!!**
- **하나 컨테이너에서 지우는 횟수가 수십개 이상쯤 부터 list가 더 빨라진다.**
- **근데 STL List도 지우는거 개느리다 웬만해선 킵해봐**

```
for (int a = 0; a < TC; a++) {
    Node* pList = nullptr;
    Node* p=nullptr;
    list_idx = 0;

    for (int i = 0; i < Narr[a]; i++) {
        p = mynew();
        p->v = i;
        p->prev = pList;
        pList = p;
    }
}
```

```
struct Node {
    int v;
    Node* prev;
} static_list[100000];
int list_idx = 0;

Node *mynew() {
    return &static_list[list_idx++];
}
int Narr[10];
```

◘ SET / PQ <int> 정렬속도 비교

10만개 int 정렬 x 2회
SET : 6ms , PQ : 7ms
결론 : SET insert, clear 작업 다 해도 PQ 전체 push pop보단 빠름.
단, 전체를 다 볼필요는 없을때나 정렬기준 다양할땐 PQ가 유리 . 느슨한 정렬로 단순 삽입은 수십배빠름

이진탐색트리 [SET] vs 힙 [PQ]
: 이진탐색트리는 완전히 크기순으로 정렬되어있으나, heap은 최상단 말고는 다소 느슨한 정렬을 한다.
-특정 키 값을 가지는 원소 검색 : 이진탐색은 좌->우 로 검색하므로 빠르게 가능하나 heap은 효율적인 방법이 없고 순회나 검색은 힙에서 쓰이지 않는다. PQ

```
for (int i = 0; i < 100000; i++) {
    ST.insert(arr[i]);
}
for (auto itr : ST) {
    in = itr;
}
ST.clear();
```

```
for (int i = 0; i < 100000; i++) {
    PQ.push(arr[i]);
}
for (int i = 0; i < 100000; i++) {
    in = PQ.top();
    PQ.pop();
}
```

◘ MAP vs Unordered_MAP 속도비교

(String 저장, 검색 4k씩)
-MAP 입력 1.5ms, 검색 1ms (mp[str1] == mp.find(str1))
-U_MAP 입력 0.7ms, 검색 0.2ms

- 절대적인 내용은 아니므로 잠깐 참고만 해라
- https://gracefulprograming.tistory.com/m/3

map.find() 와 map[ ] 의 탐색속도 차이는 무시가능

◘ Vector와 Deque 속도
Reallocation 해도 실제 둘이 속도차이 거의없는듯. 걍 vector 써라

[시간복잡도 문제]
10자 String 10만개 정렬 문제에서

1. String 10만개 heap정렬 (SET사용) : 3~4ms
2. String 10만개 hashing (unordered_map사용) : 1ms
3. **배열 arr[10만] 1000번 내용물 업데이트 : 560ms**

즉! Hash 와 heap 정렬은 10만개 수준에서는 별 문제되지 않는다.
알고리즘 자체의 시간복잡도를 개선필요!