Facebook

2020 May 29

Code in this interview is not compiled at all.

Q1. Job total time

Given a vector of job ID and an integer `time_margin`, find total processing time for all jobs if we have to run all the jobs one by one sequentially, all jobs take exactly one second, and if a job has been run before, wait for a period of integral seconds such that current execution time of a job is at least previous execution time plus `time_margin` *(exclusive)*. For example given `jobs = [A,A,B,A]` and `time_margin = 2`, the algorithm should come up with `schedule = [A,-,-,A,B,-,A]`, where dash stands for a waiting period of one second, and final output is `7`. If `time_margin = 3`, then `schedule = [A,-,-,-,A,B,-,-,A]`, output is `9`.

Solution

Python in linear time …

```
def solution(jobs, time_margin) :
      n = 0
      runtime = {}
      for job in jobs :
            if job in runtime :
                  if n < runtime[job]+wait+1 : n = runtime[job]+wait+1

            runtime[job] = n
            n = n+1
            // print snapshot here
      return n
```

Run the example and print snapshot

The snapshot at comment line when `time_margin = 2` :

```
job         latest_run        n
-           {}                0
1           {1:0}             1
1           {1:3}             4 -> wait until 0+2+1
2           {1:3,2:4}         5
1           {1:6,2:4}         7 -> wait until 3+2+1, return 7
```

The snapshot at comment line when `time_margin = 3` :

```
job         latest_run        n
-           {}                0
1           {1:0}             1
1           {1:4}             5 -> wait until 0+3+1
2           {1:4,2:5}         6
1           {1:8,2:5}         9 -> wait until 4+3+1, return 9
```

Q2. Dot product between vectors with repeated subsequences

Given a vector with many repeating subsequences of the same value, example `[1,1,1,1,6,6,6,3,3,2,2,2,1,1,1]`, how would you efficiently represent this vector in memory and calculate dot product for two vectors of this structure.

Solution

The following solution needs O(N) time and O(1) space, where *N* is number of different numbers. The iterating pattern is similar to that in question 2 of Citadel interview.

```
template<typename T> struct cell
{
      T value;
      int num; // number of occurence
};

using special_vec = std::vector<cell<double>>;
```

```
double dot_product(const repeat_vec& v0, const repeat_vec& v1)
{
	int m0 = 0; int done0 = 0;
	int m1 = 0; int done1 = 0;

	double result = 0;
	while(m0 < v0.size() && m1 < v1.size())
	{
		int size = std::min(v0[m0].num-done0, v1[m1].num-done1);
		done0 += size;
		done1 += size;
		if (done0 == v0[m0].num) { ++m0; done0 = 0; }
		if (done1 == v1[m1].num) { ++m1; done1 = 0; }
	}

	if (m0 != v0.size()) throw("size_not_match");
	if (m1 != v1.size()) throw("size_not_match");
	return result;
}
```

Q3. Finding target sum in a binary tree

Given a binary tree and integer `target`, write a function to returns `{a,b}` from different tree levels so that `a+b=target`.

Solution

The following solution involves O(N) construction of histogram and O(N) search on the histogram.

```
void construct_hist(node<int>* root, int layer, std::unordered_multimap<int,int>& hist)
{
	if (root == nullptr) return;
	hist.insert(std::make_pair(root->value, layer));
	construct_hist(root->lhs, layer+1, hist);
	construct_hist(root->rhs, layer+1, hist);
}

std::optional<std::pair<int,int>> target_sum(node<int>* root, int target)
{
	std::unordered_multimap<int,int> hist;
	construct_hist(root, 0, hist); // O(N)

	for(const auto& x : hist) // O(N)
	{
		auto range = hist.equal_range(target - x.first); // O(1)
		for(auto iter = range.first; iter!=range.second; ++iter)
		{
			if (x.second != iter->second) return std::make_optional(std::make_pair(x.first, iter->first));
		//	if (x.second != iter->second) return std::make_optional(x.first, iter->first); // compile error
		}
	}
	return std::nullopt;
}
```

Reference

How to read all values associate to a single key in `std::unordered_multimap`?

```
	std::unordered_multimap<std::string, int> map;
//	map["abc"] = 1;		// compile error, operator [] cant resolve between insert-new-entry & modify-existing-entry
//	map.insert("abc", 1);	// compile error
	map.insert(std::make_pair("abc", 1));
	map.insert(std::make_pair("def", 2));
	map.insert(std::make_pair("def", 3));
	map.insert(std::make_pair("xyz", 4));
	map.insert(std::make_pair("abc", 5));
	map.insert(std::make_pair("abc", 6));

	for(const auto& x:map)
	{
		std::cout << "\n" << x.first << " : " << x.second;
	}

	auto range = map.equal_range("abc");
	for(auto iter = range.first; iter!=range.second; ++iter)
	{
		std::cout << "\n" << iter->first << " : " << iter->second; // abc:1  abc:5  abc:6
	}
```