CS24 - Problem Solving with Computers II

Sorting and Coding Interviews

Announcements

Last week of new material! This will be the final recorded lecture, no new material next week

Previous exams on gauchospace, please come to live lecture with questions (exams or quizzes)

I will be posting a practice exam that will be similar to the final on Gradescope, highly recommend trying it.

Last week...

Heaps (STL priority queues)

Creates a “BST” that is balanced and stays sorted based off the specified comparator

Useful for sorting

Last week...

Heaps (STL priority queues)

Creates a “BST” that is balanced and stays sorted based off the specified comparator

Useful for sorting

```
void PrintPQ(priority_queue<int> toPrint){
    while(!toPrint.empty()){
        cout << toPrint.top() << ", " ;
        toPrint.pop();
    }
    cout << endl;
}
```

Last week...

Heaps (STL priority queues)

Creates a “BST” that is balanced and stays sorted based off the specified comparator

Useful for sorting

pop = Log(N)

```
void PrintPQ(priority_queue<int> toPrint){
    while(!toPrint.empty()){
        cout << toPrint.top() << ", " ;
        toPrint.pop();
    }
    cout << endl;
}
```

Sorting

Interview Question: Sort a forest of binary search trees as quickly as possible

Think about how you would implement this…

In general, start with the “brute force approach”, draw it out

Questions to ask:

- Is there a standard library that would be useful? If so ask if you are allowed to use standard library
- Is there a certain language you have to use? What data structure should we use?

Sorting

Interview Question: Sort a forest of binary search trees as quickly as possible

Think about how you would implement this…

In general, start with the "brute force approach", draw it out

Questions to ask:

Is there a standard library that would be useful? If so ask if you are allowed to use standard library

Is there a certain language you have to use? What data structure should we use?

HOW SHOULD THE TREES BE SORTED? (size, max key, min key, height, ???)

Sorting

```cpp
//Precondition: unsorted vector v of size N
//Post condition: sorted array in ascending order
template<class T>
void selectionSort(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j].size() < v[index].size()){
                index = j;
            }
        }

        T tmp = v[i];
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

```cpp
int main(){
    vector<set<int>> forest; // set is the STL standard for BSTs
    int N = 10; // number of trees
    int M = 30; // maximum number of keys in each tree
    for(int i=0; i<N; i++){
        forest.push_back(set<int>{});

        int n = rand() % M; // number of keys in tree at index i
        int Vmax = 10000; // max value of the keys
        for(int j=0; j<n; j++) // inserting keys into tree number i
            forest[i].insert( rand()%Vmax);

    }
    print( forest);
    clock_t t = clock();
    selectionSort( forest);
    t = clock() - t;

    cout << "SORTED" << endl << endl;

    print( forest);

    cout<<"Time to sort forest with  "<< N <<" trees and a max of "<<M<< " keys is "
        <<t*1000/CLOCKS_PER_SEC<<" ms"<<endl;

    return 0;

}
```

Sorting

```
Size: 11
Elements:
1478 4464 5705 5724 6334 6500 6962 8145 8467 9169 9358

Size: 1
Elements:
6827

Size: 1
Elements:
491

Size: 25
Elements:
153 292 1538 1726 1869 1942 2382 2391 3811 3902 4604 4771 4827 5436 5447
12

Size: 2
Elements:
333 7673

Size: 14
Elements:
37 778 2662 2757 2859 5141 5547 6868 7529 7644 7711 8253 8723 9741

Size: 16
```

```
Size: 1
Elements:
6827

Size: 1
Elements:
491

Size: 2
Elements:
333 7673

Size: 5
Elements:
5829 6270 6777 6924 9072

Size: 11
Elements:
1478 4464 5705 5724 6334 6500 6962 8145 8467 9169 9358

Size: 11
Elements:
1115 1673 2306 2386 2704 3977 4639 4833 5021 9658 9930

Size: 14
Elements:
```

Sorting

Analyze the runtime of the brute-force method…

Where can you trade off space complexity for time complexity?

```
Time to sort forest with  1000 trees and a max of 1000 keys is 242 ms
```

Sorting

```cpp
//Precondition: unsorted vector v of size N
//Post condition: sorted array in ascending order
template<class T>
void selectionSort(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j].size() < v[index].size()){
                index = j;
            }
        }

        T tmp = v[i];
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

N^2

```cpp
int main(){
    vector<set<int>> forest; // set is the STL standard for BSTs
    int N = 10; // number of trees
    int M = 30; // maximum number of keys in each tree
    for(int i=0; i<N; i++){
        forest.push_back(set<int>{});

        int n = rand() % M; // number of keys in tree at index i
        int Vmax = 10000; // max value of the keys
        for(int j=0; j<n; j++) // inserting keys into tree number i
            forest[i].insert( x: rand()%Vmax);

    }
    print( &: forest);
    clock_t t = clock();
    selectionSort( &: forest);
    t = clock() - t;

    cout << "SORTED" << endl << endl;

    print( &: forest);

    cout<<"Time to sort forest with  "<< N <<" trees and a max of "<<M<< " keys is "
        <<t*1000/CLOCKS_PER_SEC<<" ms"<<endl;

    return 0;

}
```

Sorting

Analyze the runtime of the brute-force method…

Where can you trade off space complexity for time complexity?

```
Time to sort forest with  1000 trees and a max of 1000 keys is 242 ms
```

```
Time to sort forest with  10000 trees and a max of 1000 keys is 2805 ms
```

Sorting

Analyze the runtime of the brute-force method…

Where can you trade off space complexity for time complexity?

```
Time to sort forest with  1000 trees and a max of 1000 keys is 242 ms
```

```
Time to sort forest with  10000 trees and a max of 1000 keys is 2805 ms
```

```
Time to sort forest with  10000 trees and a max of 10000 keys is 17004 ms
```

Sorting

```cpp
//Precondition: unsorted vector v of size N
//Post condition: sorted array in ascending order
template<class T>
void selectionSort(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j].size() < v[index].size()){
                index = j;
            }
        }

        T tmp = v[i];
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

```cpp
int main(){
    vector<set<int>> forest; // set is the STL standard for BSTs
    int N = 10; // number of trees
    int M = 30; // maximum number of keys in each tree
    for(int i=0; i<N; i++){
        forest.push_back(set<int>{});

        int n = rand() % M; // number of keys in tree at index i
        int Vmax = 10000; // max value of the keys
        for(int j=0; j<n; j++) // inserting keys into tree number i
            forest[i].insert(rand()%Vmax);

    }
    print(forest);
    clock_t t = clock();
    selectionSort(forest);
    t = clock() - t;

    cout << "SORTED" << endl << endl;

    print(forest);

    cout<<"Time to sort forest with  "<< N <<" trees and a max of "<<M<< " keys is "
        <<t*1000/CLOCKS_PER_SEC<<" ms"<<endl;

    return 0;

}
```

Sorting

What if we rearrange pointers instead of BSTs?

Sorting

What if we rearrange pointers instead of BSTs?

```
vector<set<int>*> GenerateIntSetPointerVector(int num_trees, int num_keys, int keyMax){
    vector<set<int>*> forest; // set is the STL standard for BSTs

    for(int i=0; i<num_trees; i++){
        forest.push_back(new set<int>{}); // Using new pushes a pointer instead of a set

        int n = rand() % num_keys; // number of keys in tree at index i

        for(int j=0; j<n; j++) // inserting keys into tree number i
            forest[i]->insert( x: rand()%keyMax);

    }
    return forest;
}
```

Sorting

What if we rearrange pointers instead of BSTs?

```cpp
vector<set<int>*> GenerateIntSetPointerVector(int num_trees, int num_keys, int keyMax){
```

```cpp
template<class T>
void selectionSortPointer(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j]->size() < v[index]->size())
                index = j;
        }

        T tmp = v[i]; // all of this can now be done in constant time thanks to not having to write new BSTs
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

Sorting

What if we rearrange pointers instead of BSTs?

```
vector<set<int>*> GenerateIntSetPointerVector(int num_trees, int num_keys, int keyMax){
    ve
    fo
    }
    re
}
```

```
template<class T>
void selectionSortPointer(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j]->size() < v[index]->size())
                index = j;
        }

        T tmp = v[i]; // all of this can now be done in constant time thanks to not having to write new BSTs
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

```
Time to sort forest with  10000 trees and a max of 10000 keys is 515 ms
```

Sorting

What if we rearrange pointers instead of BSTs?

```
vector<set<int>*> GenerateIntSetPointerVector(int num_trees, int num_keys, int keyMax){
```

```
template<class T>
void selectionSortPointer(vector<T>& v){
    int N = v.size();
    for(int i =0; i < N; i++){
        int index=i;
        for(int j = i+1; j< N;j++){
            if(v[j]->size() < v[index]->size())
                index = j;
        }

        T tmp = v[i]; // all of this can now be done in constant time thanks to not having to write new BSTs
        v[i] = v[index];
        v[index]=tmp;
    }
}
```

```
Time to sort forest with  10000 trees and a max of 10000 keys is 515 ms
```

*** Still N^2 ***

Sorting

What data structure have we learned that can sort data in less than N^2 time?

Sorting

What data structure have we learned that can sort data in less than N^2 time?

Heaps!

Sorting

Heaps constantly keep track of the max/min value and can pop in log(n)

Total time for sorting can be simplified to O(Nlog(N))

```
template<class T, class cmpClass>
void heapSort(vector<T>& v){
    priority_queue<T,vector<T>,cmpClass> pq;
    int N = v.size();
    for(int i =0; i < N; i++){
        pq.push(v[i]);
    } // Nlog(N)
    int i=0;
    while(!pq.empty()){
        v[i]=pq.top(); //O(1)
        pq.pop(); //O(log N)
        i++;// O(1)
    }//NlogN
    // Total = O(NlogN)
}
```

Sorting

Heaps constantly keep track of the max/min value and can pop in log(n)

Total time for sorting can be simplified to O(Nlog(N))

```
template<class T, class cmpClass>
void heapSort(vector<T>& v){
    priority_queue<T,vector<T>,cmpClass> pq;
    int N = v.size();
    for(int i =0; i < N; i++){
        pq.push(v[i]);
    } // Nlog(N)
    int i=0;
    while(!pq.empty()){
        v[i]=pq.top(); //O(1)
        pq.pop(); //O(log N)
        i++;// O(1)
    }//NlogN
    // Total = O(NlogN)
}
```

```
Time to sort forest with  10000 trees and a max of 10000 keys is 7 ms
```

Sorting

Heaps constantly keep track of the max/min value and can pop in log(n)

Total time for sorting can be simplified to O(Nlog(N))

```
template<class T, class cmpClass>
void heapSort(vector<T>& v){
    priority_queue<T,vector<T>,cmpClass> pq;
    int N = v.size();
    for(int i =0; i < N; i++){
        pq.push(v[i]);
    } // Nlog(N)
    int i=0;
    while(!pq.empty()){
        v[i]=pq.top(); //O(1)
        pq.pop(); //O(log N)
        i++;// O(1)
    }//NlogN
    // Total = O(NlogN)
}
```

2429 times faster!!

```
Time to sort forest with  10000 trees and a max of 10000 keys is 7 ms
```

Other Problem Solving/Interview Tips

Solve an easier version of the problem

Draw pictures

Use pseudocode

Worry about exactness once the main problem is solved

Follow proper naming conventions

Think about edge cases

That’s it!

No more new material!!
Please come to live lecture next week with questions!