CS218, Winter 2024, Due: 11:59pm, Friday 02/02, 2024
Assignment #2 [1] Training Programming Due: 11:59pm, Friday 01/26, 2024
Name: Student ID:

---

1 (3pts) Deadlines, again!

1. (0.2pts) So we have n = 3 Assume we apply a greedy algorithm of sorting the assignments by their deadline and we have a tie breaker rule that if 2 assignments have the same day submission, choose the one having more points. Even using this strategy it will not work. The following case is a counter example for n = 3

   (a) hw1 - deadline day 2 points = 5

   (b) hw2 - deadline day 2 points = 7

   (c) hw3 - deadline day 3 points = 8

   Now greedy will only choose one out of hw1 and hw2 and then will choose hw3. But the optimal solution is to choose hw1 on day 1, hw2 on day2 and hw3 on day 3

2. (0.2pts)

   Same example as above can be used to disprove this greedy approach also. As this approach will do hw3 first then hw2. But ideal will be like I described above.

3. (0.5pts) Assume there is an optimal homework sequencing for n homeworks and each having deadline d_1 to d_n. Assume that homework i, having the profit p_i is the maximum profit one and it has a deadline d_j. Now in the optimal solution, instead of finishing this homework some other homework k, having the profit p_k is being done on the day d_j. If we replace it with assignment i, we get an increase by $p_i - p_k$ and the rest of the optimal order remains unchanged. Hence the max profit one will always be inside the optimal order.

4. (0.5pts)

   - If we have n assignments, each having a different deadline then we can just place them on the respective day of deadline and it will be solved as $d <= n$. Problem arises when assignments having different costs have the same deadline. We do not have the case when same cost and same deadline exists as assumption is that costs are unique.
   - Assume we sort the assignment profit in descending order.
   - Loop through the array.
   - First item is the largest profit which will always be in the optimal solution. Better to put it at on the day of its deadline.
   - Assume at a point when profit assignment p_j has deadline d_j. If there is no space in d_j then go back and see if you can find an empty day where you can add p_j.
   - If we cannot find the day at all for pj from d_1 to d_j then it means all are occupied by assignments which have profit greater than p_j so replacing any assignment with p_j will decrease the cost hence better to just drop p_j.

5. (0.3pts) Applying this for the given example

   - Sort them according to the profit we get

[1] Some of the problems are adapted from existing problems from online sources. Thanks to the original authors.

- (15, 5) (12, 3) (10, 13) (8, 2) (7, 2) (5, 5) (4, 4) (1, 7)
- Take 15, put it at day 5
- Take 12 put it at day 3
- Take 10 put it at day 13
- Take 8 put it at day 2
- Take 7 it should go at place 2 but it is filled so go back and check empty which is day 1
- Take 5 it should go at place 5 but it is filled with 12 so go back and check the empty day which is day 4.
- Take 1 put it at day 7
- 4 is ignored totally

Total profit hence is $15 + 12 + 10 + 8 + 7 + 5 + 1 = 58$

6. (0.8pts) Here what i am doing is I am going in descending order, taking the highest profit first and trying to fit it at the day of its deadline, if it does not then I go back and fit in some other day before its deadline.

   If one starts with highest then it will fit on the day of its deadline as it is the starting element. Now the problem modifies to the (profit of the highest + optimal solution of the remaining). Greedy choice here is optimal as proved above in part 4.

7. (0.5pts)

   - Now if we start looking the problen from the base case, we can say that the highest priority one is in the output + the solution for the remaining set of assignments.
   - Now in this new set we again will have this set's respective top priority assignment, so it will definitely be in the output. Then the answer will be (highest assignment at some day) + (second highest assignment at other day) + priority for remaining set of jobs.
   - This is different from the "highest priority first" in the above part 2 is that it the above greedy choice always fit the highest job to a list of days starting from d_1 to d_n, this order need not to be true.
   - So my greedy algorithm above chooses the highest priority job first then assigns them a specific day in a decreasing manner.

2 (1.5pts) Don't be late!

- Assume lift time is A, stairs time is B, what we have here is the condition $A < B$.
- So for time t we get the ratio as $(t + A)/(t + B$.
- Problem arises in choosing a value of t.
- Choosing B - A should do here because then the ratio becomes decreasing and bounded $2 - A/B$. No matter what the value of A or B as long as inequality in first point holds, this ratio will be bounded.

3 (1.5pts) Time Turner

We maintain 2 lists, one which has courses which will be taken in the first iteration, the other having those courses taken after using the time turner.

- After sorting the course list based upon end time of a course and pushing the first course into list 1 (which is the first iteration list).
- We maintain 2 variables denoting end time of the latest course in each list.
- If we encounter a new course while we loop the list, we see its compatibility with the 2 lists and put it accordingly, if its compatible with both.
- Then we make the greedy choice, that it goes with the course in which the time gap between ending of the previous course and this current course is less.

Submission id 244215650

4 (1.5pts) Share Candies

5 (1pts) The Security System

This code I was not able to submit on time as I thought the deadline for contest will extend in the same way as grace days provided. I however devised a greedy approach:

- Sort furniture based upon end point, if 2 furniture have same end point then choose the one which has lesser length.
- What I mapped this question into my mind, is that we have some n overlapping lectures and k classrooms, we have to choose the maximum subset among the n lectures such that the depth of the subset is k.
- Base case of k = 0 one can easily just return n as the answer.
- For $k >= 1$
- Create a priority queue. Key is ending point of the furniture, if same then the starting point.
- Make a remove var, initialize to 0, this is our final answer.
- Loop through the sorted array, put the first element in the queue. Now your count is 1.
- When you encounter any next element, see if it intersects with the top element of the priority queue, if yes then increase count.
- If count is greater than k, meaning this element is one of those which we have to remove, so ignore this element and increase the value of remove.
- If not then push it in the queue. If no intersection, then remove the top element of the queue and push this element in the queue.
- When the loop is finished output remove.

Here we are making the greedy choice that furnitures having greater length will be checked after the small length ones.

Since I cannot check the code now, I am below submitting my code file which you can run on the checker as it is.

Code looks long because of my commented out part. Its C++20

```
#include<iostream>
#include<vector>
#include<tuple>
#include<queue>

using namespace std;

typedef tuple<long long int, long long int, long long int, long long int, long long int, lo
typedef tuple<long long int, long long int> classroom;


int countk=0;
int removek=0;
int printsofa(sofa elem)
{
    cout << "Element is " << get<0>(elem) << " " << get<2>(elem) << endl;
    return 0;
}

int printclass(classroom elem)
{
    cout << "Element is " << get<0>(elem) << " " << get<1>(elem) << endl;
    return 0;
}

int compare(sofa lhs, sofa rhs)
{
    if(get<2>(lhs) < get<2>(rhs))
    {
        return 1;
    }
    else if(get<2>(lhs) == get<2>(rhs))
    {
        return (get<4>(lhs) < get<4>(rhs));
    }
    else
    {
        return 0;
    }
}
int intersect(sofa elem, classroom b)
{
    classroom a = make_tuple(get<0>(elem), get<2>(elem));
    if(!((get<0>(a) > get<1>(b)) || (get<1>(a) < get<0>(b))))
    {
        // cout << "Intersect found" << endl;
        // printclass(a);
        // printclass(b);
        return 1;
    }
    else
```

```
    {
        return 0;
    }
}
int main()
{
    long long int l, n, k;
    scanf("%lld %lld %lld", &l, &n, &k);
    long long int a[l];
    long long int obstacles = 0;
    for(long long int i=0; i<n; i++)
    {
        a[i] = 0;
    }
    // priority_queue <sofa, vector<sofa>, decltype(cmp) > pqueue;
    auto cmp = [](const classroom lhs, const classroom rhs)
    {
        if(get<1>(lhs) < get<1>(rhs))
        {
            return 0;
        }
        else if(get<1>(lhs) == get<1>(rhs))
        {
            if(get<0>(lhs) < get<1>(rhs))
            {
                return 0;
            }
        }
        else
        {
            return 1;
        }
    };
    priority_queue <classroom, vector<classroom>, decltype(cmp)> pqueue;
    vector<sofa> t;
    for(long long int i=0; i<n; i++)
    {
        long long int a, b, c, d, obstructions;
        scanf("%lld %lld %lld %lld", &a, &b, &c, &d);
        obstructions = c - a + 1;
        sofa one = make_tuple(a, b, c, d, obstructions, 0);
        // pqueue.push(one);
        t.push_back(one);
    }
    sort(t.begin(), t.end(), compare);
    // cout << "after sorting" << endl;
    // for(long long int i=0; i<t.size(); i++)
    // {
    //     printsofa(t[i]);
    // }
    if(k == 0)
```

```
    {
        cout << n << endl;
        return 0;
    }
    for(long long int i=0; i<t.size(); i++)
    {
        // cout << "Current element ";
        // printsofa(t[i]);
        if(pqueue.size() == 0)
        {
            pqueue.push(make_tuple(get<0>(t[i]), get<2>(t[i])));
            // cout << "adding because queue is empty " << endl;
            countk++;
        }
        else
        {
            // cout << "Current top ";
            // printclass(pqueue.top());
            classroom lowest = pqueue.top();
            if(intersect(t[i], lowest))
            {
                countk++;
                if(countk > k)
                {
                    // cout << " Ignoring ";
                    // printsofa(t[i]);
                    removek++;
                    countk--;
                }
                else
                {
                    classroom new_one = make_tuple(get<0>(t[i]), get<2>(t[i]));
                    pqueue.push(new_one);
                }
            }
            else
            {
                // cout << "Removing ";
                // printclass(pqueue.top());
                pqueue.pop();
                classroom new_one = make_tuple(get<0>(t[i]), get<2>(t[i]));
                pqueue.push(new_one);
            }
        }
    }
    cout << removek << endl;
    return 0;
}
```