CS-431 PROGRAMMING LANGUAGES LAB ASSIGNMENT-3

-MAYANK WADHWANI (170101038)

a) Algorithm that I used to solve the problem

It was observed after going through the constraints, that we could subdivide our problem into two to reduce/prune the search space. This is because that the garden and balcony are always fixed to be 1 and are always independent of the other 4 rooms. So if we break the 6 rooms into 2 and 4 independent tasks, then our search space gets drastically improved.

So the problem has now been broken. We simply compute the cross product of first 4 rooms and next 2 rooms(garden and balcony) separately. To compute the cross product of 4 rooms, we simply compute cross product of 2 rooms and then merge these 2 cross products to get 1 net cross product. One ***optimization*** *that has been done is, when we merge these 2 cross products, then the elements which has the same area are removed. This is because our task is to minimize the unused space. We do not have to maximize on the dimensions. Till the constraints that bedroom size is greater than or equal to kitchen and kitchen greater than equal to bathroom is satisfied, we are good to go.*

*More specifically, if say we have two possible room architectures with dimensions (10*10 and 10*4) and (10*7 and 10*7), we can remove one of them since their areas are same (140). This helps in significant speedup thus reducing the search space.*

Once this is done, we finally merge the two cross products (one for the 4 rooms, one for 2 rooms) to get a final cross product which is also pruned. We then search this cross product to get a possible architecture with the highest values of total area. We finally compute the unused space and print the values.

Psuedo-Code

```
design area bedroom_count halls_count:
    A = all possible dimensions for bedrooms
    B = all possible dimensions for halls
    compute and prune cross_product for A and B storing the area
    C = all possible dimensions for kitchen
    D = all possible dimensions for bathroom
    compute and prune cross_product for C and D storing the area
    Compute and prune the net cross product for A,B , C, D
    E = all possible dimensions for garden
    F = all possible dimensions for balcony
    Compute and prune the cross product for E, F
    Merge all cross products to get a final pruned cross product for A, B, C, D, E, F
    Find the element with the highest area in the cross product
    Print all attributes of the element
```

b) How many functions did you use?

In total, I have used 14 functions in my program to get the compute. These functions have been modularized to make the code robust to changes.

1 function is the main driver function (design) which the user will call

1 function is to print the maximum element of the final set,

1 function is to do linear search and find the maximum element of the set

1 function is to prune the search space given a set as input,

4 functions is to compute cross product for 6 rooms and 4 rooms (2+2)

1 function is to iterate over all possible values of count of kitchen and find cross products with bathrooms,

2 function is to generate a list of all possible dimensions given start and end as input

1 function is to compute cross product for 2 given lists of possible dimensions

2 functions to insert unique elements in the list while checking if the element is already present

c) Are all those pure?

No, not all the functions of the program are pure. Although 12 of the 14 functions are pure. A pure function is a type of function which always output the same result with the same input. But the functions involving the use of an IO type constructor implies that a function is impure. So 2 of our functions are impure, mainly the main driver function and the print properties function that displays IO().

(Source-> http://www-cs-students.stanford.edu/~blynn/haskell/io.html)

d) If not, why?

We could have used only pure functions in our program to maybe return a list of areas that denotes the areas occupied by each rooms. But since it was required in the question to print all the attributes, we used the IO constructor, and if we use the IO constructor, this implies the function is impure. So in short we cannot solve the problem using pure functions only. If we still want to solve it, then we will have to only perhaps return the final list of values and not print it. Then we would be able to make all functions pure.

Short Notes on lazy evaluation and advantages of Haskell

a) Do you think the *lazy* evaluation feature of Haskell can be exploited for better performance in the solutions to the assignments? If so, which solution(s) and how?

Yes, the lazy evaluation feature can be exploited for a better performance in the solutions. While using lazy evaluations, we compute only when required. Like we can have infinite lists in Haskell which is not possible in languages like C++. Haskell only allocates space which is required. So if we declare an infinite list and then write take 20, then only space for these 20 elements will be allocated. This is how the lazy feature of Haskell can be exploited, we can declare very large data structures (lists in our case) and still not worry about the implementation.

Yes, this has been used in the third question where we had to generate the lists for all possible dimensions. We can optimize our solution by using zip and filters which work directly on lists. By doing this, we could have optimized to get faster results and better memory usage since many redundant areas would not be initialized, also in cross product computation, we can lazily find only those elements which satisfy the given constraints on the dimensions. In such a manner, we can exploit lazy evaluation of Haskell for better performance.

b) We can solve the problems using any imperative language as well. Do you find any advantage of using Haskell for these problems (w.r.t the property of lack of side effect)? If your answer is no, elaborate on why not?

Indeed, Haskell serves a lot of advantages here. We say that some function has a side effect if it modifies some variables outside its own scope ie. it modifies global variables. This can occur if we do IO or change some values in a list.

The main problem of side effects is that it makes functions unpredictable. Unpredictable functions are sometimes quite hard to debug. We are not always sure of the output we would get. This indeterministic property is many-a-times undesirable.

Haskell has the property of having lack of side effects, if we give an input, we can be sure of the output. This makes it easy to debug if we are stuck and we can be sure of the determinism of the outputs we get on some input. And hence I feel Haskell provided some advantages in solving the assignments by ensuring lack of side effects.

END OF REPORT