Dynamo

A Programming language to model Dynamic Programming algorithms for optimization problems.

Team:
Raghavan Muthuregunathan rm2903
Pradeep Dasigi pd2359
Abhinav Saini as3906
Srilekha V. K. sv2344
Archana Balakrishnan ab3416

Objective of Dynamo:
Dynamic Programming is a very useful method for solving complex problems by breaking them down into smaller sub-problem. Essentially, any problem that could use this approach has the following characteristics:
1) Optimal Substructure
2) Overlapping Sub-problems
However, it is not intuitive to implement such algorithms in the general purpose high level languages.
The goal of Dynamo is to make the representation of the algorithm as intuitive as writing its core mathematical equations.

Keywords:
We believe that a dynamic programming algorithm can easily be represented in the following notation. Below is a subset of keywords of the language and their description

Data_Init represents the data structure from where the data is picked up in building the bottom up

Base_Case represents a block that has the base case conditions in the format
<output>
<conditional>;

Recursive represent the recursive substructure of the DP problem, in the same format as the Base_Case.

iterate links an iterative variable to a data structure that it iterates on

The following code snippet explains the language by taking Longest Common Subsequence (LCS) problem as an example.

Code Snippet

The equation below represents the recursive substructure of the Longest common subsequence

problem. (image taken from wikipedia)

$$LCS\left(X_i, Y_j\right) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ \left(LCS\left(X_{i-1}, Y_{j-1}\right), x_i\right) & \text{if } x_i = y_j \\ \text{longest}\left(LCS\left(X_i, Y_{j-1}\right), LCS\left(X_{i-1}, Y_j\right)\right) & \text{if } x_i \neq y_j \end{cases}$$

```
DP LCS
{
  Data_Init Sequence1
  Data_Init Sequence2
  //DP_Component Array
 iterate i on Sequence1
 iterate j on Sequence2
 Base_Case:
        null
        if i = 0;
        null
        if j = 0;
 Recursive:
        LCS(i-1,j-1)+Sequence1[i]
        if Sequence1[i] == Sequence2[j];
        max(LCS(i-1,j), LCS(i,j-1))
        if Sequence1[i] != Sequence2[j];
}
```

Checking for correctness of Recurrence relation:

The compiler will check for errors in the recursive substructure defined. For example, a cyclic dependency like *f(x) = f(x+1)+f(x-1)* will be recognized as an error since *f(x)* and *f(x+1)* are mutually dependent on each other, and either one has to have a value for computing the other.

Possible Challenges:

* Understanding complex recurrences
* Scalability when the recurrence equation involves more than three variables