IOI Training Week 5
DP on Trees and DAGs

Tim Dumol

Contents

1 Trees and Directed Acyclic Graphs (DAGs)

Recall from the graphs section the following definitions (if any of them feel vague, then please ask the trainer or just search for the terms):

Definition 1 (Graph)**.** A graph, G, consists of a set of v vertices (or nodes) and a set of e edges, which connect pairs of vertices.

Definition 2 (Directed Graph)**.** A directed graph, G, is a graph whose edges have a start and an end (their edges have a direction, or orientation).

Definition 3 (Chain (or Walk))**.** A chain is an alternating sequence of vertices and edges, such that v_i and v_{i+1} are connected by edge e_i.

Definition 4 (Path)**.** A (simple) path is a chain that does not have repeated vertices.

Definition 5 ((Directed) Cycle)**.** A directed cycle is a chain of vertices and edges that start and end at the same vertex.

Definition 6 (Directed Acyclic Graph)**.** A directed acyclic graph (DAG) is a directed graph which does not have any cycles.

Definition 7 (Connected Graph)**.** A connected graph is a graph such that there exists a path between every pair of vertices.

Definition 8 ((Unrooted) Tree)**.** A (unrooted) tree is a connected (undirected) graph that has v vertices and $v - 1$ edges. Equivalently, a tree is a graph such that there exists exactly one path between every pair of vertices. Equivalently, a tree is a connected acyclic undirected graph.

The vertices of the tree with connectivity 1 are called the *leaves* of the tree.

Definition 9 (Orientation of a graph)**.** An orientation of an undirected graph G is a directed graph G' formed by assigning a direction to each edge of the graph G.

Definition 10 (Rooted Tree)**.** A rooted tree is an orientation of an unrooted tree, such that there exists a vertex r, called the *root* of the tree, such that there exists a path from the root to every other vertex of the tree.

Colloquially, you can imagine getting a vertex of an unrooted tree, and then pulling it up and dangling the rest of the tree from it.

2 Dynamic Programming (DP) on Trees and DAGs

The structure of DAGs and trees make them particularly amenable to dynamic programming. This can be seen most easily through an example. And since I'm a mathy guy, we'll start with a definition (actually, several definitions):

Definition 11 (Independent Set)**.** An independent set of a graph is a set of vertices in a graph, no two of which are independent.

Definition 12 (Maximum independent set)**.** A maximum independent set of a graph is an independent set of the largest possible size for the given graph.

(Note: this is distinct from a *maximal* independent set, which is an independent set to which you cannot add any more vertices. In general, a *maximal* thing is something to which you can't add to, while a *maximum* thing is the largest possible of that thing. All *maximum* sets are *maximal*, but not vice versa.)

Definition 13 (Maximum-weight independent set)**.** A maximum-weight independent set of a weighted graph (where weights are attached to each nvertex) is an independent set of the largest possible sum of weights in the given graph.

In general, finding a maximum independent set (and by extension, the maximum-weight independent set) of a graph is an *NP-hard* problem (there is no known sub-exponential algorithm to find it). However, the simple structure of a tree makes it simple to find the maximum independent set of a tree[1].

The trick of doing DP on a DAG (and a rooted tree is a DAG) is, as with all DP, to decompose the problem into subproblems, solve the subproblems, and then put the solutions to the subproblems together. The difference with DP on a tree/DAG is that the dependency structure of your solution has been handily been given to you.[2] In particular, your DP solution will often be of the form:

```
f(vertex) := g(f(child) for child in children(vertex))
```

(substituting `neighbors(v)` for DAGs). For example, to find the maximum-weight independent set of a tree, simply root the tree[3], and try to break it down into subproblems. In this case, we can exhaustively break it down into two cases: either we pick the current vertex (in which case we can't choose its children), or we don't (in which case we can). In code:

```
typedef vector<int> EdgeList;
typedef vector<EdgeList> AdjList;

constexpr int N = 1024;

AdjList graph;
int weights[N];
// assume graph and weights are initialized
int ans[N];
fill(ans, ans+N, -1);

int f(int v) {
  if (ans[v] != -1) return ans[v];
  // choose children to put into set, so you can't choose this vertex
  int childrenSum = 0;
  for (auto it = graph[v].cbegin(); it != graph[v].cend(); ++it) {
    childrenSum += f(*it);
  }
  // choose this vertex, so you can't choose children
```

[1] There are many other classes of graphs for which it is simple to find the maximum independent set of.

[2] In fact, you may find that the call tree of your solution (which is also a DAG), happens to be pretty similar to the DAG in your problem.

[3] in this case, we can take an arbitrary node and orient all other edges away from it–in other cases, you may want to minimize the height of the tree, which you can do by finding the *center* of the graph, and setting that as the root)

```
  int selfSum = weights[v];
  for (auto it = graph[v].cbegin(); it != graph[v].cend(); ++it) {
    int child = *it;
    for (auto it2 = graph[child].cbegin(); it2 != graph[child].cend(); ++it2) {
      selfSum += f(*it2);
    }
  }
  return (ans[v] = max(childrenSum, selfSum));
}

printf("%d\n", f(root));
```

The same reasoning applies to DAGs, of course, except that instead you may have multiple starting points.

Now that the theory's done–the best way to learn DP is by practicing. So without further ado, the problems.

3 Problems

1. Unidirectional TSP (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=52)
2. Spreadsheets (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=132)
3. Not So Mobile (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=780)
4. Walk Through the Forest (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1858)
5. The Queue (https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=391&page=show_problem&problem=3003)
6. Alternative Aborescence (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=2282)
7. Sultan's Chandelier (https://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=2535)
8. Matrix (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1299)
9. Dragster (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=78&page=show_problem&problem=2727)
10. Optimal Cut (https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=117&page=show_problem&problem=2882)

3.1 Bonus Problems

These problems are ungraded.

1. IT Restaurants (http://codeforces.com/problemset/problem/212/E)
2. Shaass the Great (http://codeforces.com/problemset/problem/294/E)
3. Ostap and Tree (http://codeforces.com/problemset/problem/735/E)

4 References

1. DP on Trees Tutorial — Codeforces (http://codeforces.com/blog/entry/20935)