Exercício 24.2–3

Leonardo de Paula Rosa Piga

Enunciado

The PERT chart formulation given above is somewhat unnatural. It would be more natural for vertices to represent jobs and edges to represent sequencing constraints: that is edge (u, v) would indicate that job u must be performed before job v. Weights would then be assigned to vertices, not edges. Modify the `DAG-SHORTEST-PATHS` procedure so that it finds a longest path in a directed acyclic graph with weighted vertices in linear time.

Resolução

A solução necessita que o grafo de entrada seja transformado em um grafo de PERT em que as arestas representam atividades, como definido originalmente.

Para isto, cria-se um grafo $G'(V', E')$ a partir do grafo de entrada G. Adiciona-se um novo vértice fonte s a V' de maneira que $V' = V \cup \{s\}$. Todas as arestas de E estão em E' e E' também inclui uma aresta (s, v) para cada vértice $v \in V$ que tenha grau de entrada zero em G. Desta maneira, o único vértice com grau de entrada zero em G' é a nova fonte s. Em seguida, para cada aresta (u, v) de G' atribui-se peso correspondente ao valor de v do grafo original. No novo grafo, $|V'| = |V| + 1$ e $|E'| = O(|V| + |E|)$.

Os algoritmos abaixo mostram a implementação desta ideia. Foram reimplementadas as funções `INITIALIZE-SINGLE-SOURCE` trocando "∞" por "$-\infty$", a comparação "$>$" de `RELAX` foi substituída por "$<$" e implementou-se o procedimento `TO-UNNATURAL-PERT` que converte um grafo de PERT em que os **vértices** representam tarefas em um grafo de PERT em que as **arestas** as representam.

Algoritmo 1 INITIALIZE-SINGLE-SOURCE(G, s)

```
1: for cada vértice v ∈ V[G] do
2:     d[v] ← −∞
3:     π[v] ← NIL
4: end for
5: d[s] ← 0
```

O procedimento `TO-UNNATURAL-PERT` gasta tempo $\Theta(V + E)$, que é a mesma complexidade da ordenação topológica, logo, a complexidade final total continua sendo $\Theta(V + E)$.

Algoritmo 2 RELAX(u, v)

```
1: if d[v] < d[v] + w(u, v) then
2:     d[v] ← d[u] + w(u, v)
3:     π[v] ← u
4: end if
```

Algoritmo 3 TO-UNNATURAL-PERT(G)

```
1: G' ← cópia de G
2: s ← novo vértice
3: V' ← V ∪ {s}
4: for cada vértice v de grau zero em G' do
5:     adiciona aresta (s, v)
6: end for
7: for cada aresta (u, v) de G' do
8:     w(u, v) = w[v]
9: end for
10: return s, G'
```

Algoritmo 4 DAG-LONGEST-PATHS(G)

```
1: s, G' ← TO-UNNATURAL-PERT(G)
2: faça uma ordenação topológica nos vértices de G'
3: INITIALIZE-SINGLE-SOURCE(G', s)
4: for cada vértice u, escolhido na ordem topológica do
5:     for cada vértice v ∈ Adj[u] do
6:         RELAX(u,v)
7:     end for
8: end for
```