SuperCon 2001
予選プログラム解説

近藤 健一　　三井 健太郎　　木下 峻一

2001 年 8 月 12 日

1　今回の問題について

今回の問題は恐らく行列連積 (Matrix Chain Multiplication) の問題を基にして作ったのではないでしょうか。行列連積の場合、たとえば $a \times b$ の行列と $b \times c$ の行列を掛けるときのコストはスカラの乗算回数 $a \times b \times c$ で評価しますが、これを加算 $a + b + c$ にしたのが今回の問題となります。

このことには、MIT の *Introduction to Algorithms* と言う本で行列連積の項目を見付けるまで気づきませんでした。他にも何か利用できることが載っているかと思ったのですが、動的計画法も知らないうちに使っていたので、アルゴリズムの確証を得られた以外に特に収獲はありませんでした。

今回一番心配なのは、ごく当前の理論的展開しかしていない、つまりこの問題に特化したような枝切り等ができていないことです。その分コーディング上のチューニングはかなりやりましたが、やはり心配です。

2　基本的なアルゴリズム

最も基本となるアルゴリズムは再帰を使ったもので、演算子の全ての実行順序を調べます。

まず

$$\begin{aligned} \text{ペア } P_k &: (T_k, T_{k+1}) \\ \text{列 } S_{h,l} &: P_h @ P_{h+1} @ \ldots @ P_{h+l-1} \\ \text{コスト } C_{h,l}\ (l > 1) &: S_{h,l}\text{の最小計算コスト} \end{aligned}$$

とおき、初めに与えられた列を $S_{0,n}$ とします。

プログラムでは $C_{h,l}$ を求めるために、h, l を引数として関数 `calc_cost` を呼び出します。もし $l == 2$ ならば、単純にコスト $C_{h,2} = T_h + T_{h+1} + T_{h+2}$ を返します。それ以外の場合は $S_{h,l}$ を $S_{h,k}$ と $S_{h+k,l-k}$ とに分割して $C_{h,l}$ を

求めます。まず一時変数 $cost_k$ を次のようにおきます。

$$\begin{aligned} cost_1 &= C_{h+1,l-1} + T_h + T_{h+1} + T_{h+l} \\ cost_k &= C_{h,k} + C_{h+k,l-k} + T_h + T_{h+k} + T_{h+l} \ (1 < k < l-1) \\ cost_{l-1} &= C_{h,l-1} + T_h + T_{h+l-1} + T_{h+l} \end{aligned}$$

このとき $1 < k < l$, $1 < l-k < l$ が成り立つので、再帰的に `calc_cost` を呼び出すことによって $C_{h,k}, C_{h+k,l-k}$ の値が求まります。次に $cost_k$ の最小値、つまり $C_{h,l}$ を求めてそれを返します。

このアルゴリズムが正しい理由を簡単に述べます。二項演算子の括弧づけ (実行順序) の問題は、葉が被演算子、葉以外の節が演算子からなる二進木の構築に他なりません。上述の関数は、与えられた列からある演算子を選んで根にし、そこに左右の部分木を再帰的に構築して繋げるという操作を全ての演算子について行っていることになります。つまり全ての二進木を構築及び評価しているということになるので、このアルゴリズムは正しいと言えます。

3 高速化

先に述べたアルゴリズムの実行時間は大雑把に言って $O(n!)$ なので、とてつもなく時間がかかってしまいます。

少し考えると、ある h, l について $C_{h,l}$ を計算して返すことが何回もあることが分かります。しかも最小コストは毎回一定なので、この値を配列に格納しておけば非常に時間が節約できます。

そこでトップダウン的アルゴリズムから、次のようなボトムアップ的アルゴリズムに変更します。実行時間はだいたい $O(n^3)$ です。

1. $C_{h,2}$ を求めます。この場合は単にペアの数字を足すだけになります。
2. $C_{h,3}$ を求めます。前の結果を使って

$$\begin{aligned} C_{h,3} &= min(C_{h+1,2} + T_h + T_{h+1} + T_{h+3}, \\ &\qquad C_{h,2} + T_h + T_{h+2} + T_{h+3}) \end{aligned}$$

   として求めます。
3. $C_{h,4}$ を求めます。この場合も前の結果を使って

$$\begin{aligned} C_{h,4} &= min(C_{h+1,3} + T_h + T_{h+1} + T_{h+4}, \\ &\qquad C_{h,2} + C_{h+2,2} + T_h + T_{h+1} + T_{h+3}, \\ &\qquad C_{h,3} + T_h + T_{h+3} + T_{h+4}) \end{aligned}$$

   として求めます。
4. 以下同様にして $C_{h,l}$ を l が小さい方から求めます。この場合もそれまでに計算した $C_{h,m}$ $(m < l)$ の値を利用します。

5. $l == n$ のときの計算を終えた時点で、$C_{0,n}$ が求めるコストとなっています。

4 チューニング

あとはプログラムのチューニングによって実行速度をあげています。

1. P_k ではなく T_k を配列に格納しています。
2. $C_{h,l}$ を配列に格納していますが、`calc_cost` の中の加算を減らすため余分に T_h を加えています。
3. 配列の添字計算が遅いので、ポインタとテーブルを組み合わせて使っています。
4. その他細かい点がいくつか。

5 変更履歴

2001 年 6 月 26 日：暫定版。

2001 年 6 月 27 日：予選応募版。S や C 等の記号を使って簡潔にした。

2001 年 8 月 11 日：公開にあわせて見直し。文章を簡潔にして、添字の間違いを修正した。

2001 年 8 月 12 日：細かい変更。