アルゴリズムノート

Masanari Kimura

1　ニュートン法

ニュートン法は，ある方程式 $f(x) = 0$ の実数解を求める手法のうちの一つ．手順としては，

1. 初期値 x_0 を決める．
2. $f(x)$ の x_n の接線を求める．接線の方程式は $y - f(x_n) = \frac{d}{dx} f(x_n)(x - x_n)$.
3. 求めた接線と y 軸との交点を求め，それを x_{n+1} とする．つまり，$0 - f(x_n) = \frac{d}{dx}(x_n)(x_{n+1} - x_n) \rightarrow x_{n+1} = -\frac{f(x_n)}{f'(x_n)}$.
4. $2 \sim 3$ を繰り返す．

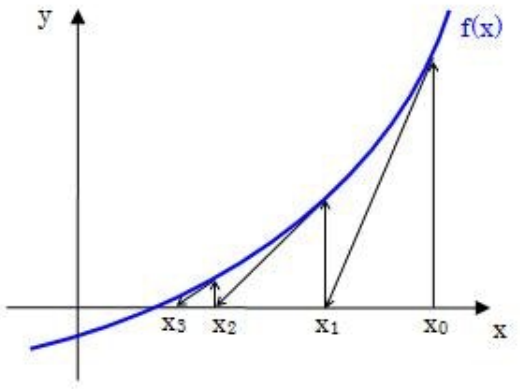

Figure 1: ニュートン法

1.1 2 の平方根を求める

このとき，方程式は $f(x) = x^2 - 2$.
　計算機上で実現する際には，関数の微分は数値微分の近似を用いればいい（例えば 2 点近似など）.

```
double f(double x) {
  return pow(x, 2) - 2;
}

double df(double x) {
  return (f(x+1e-6) - f(x)) / 1e-6;
}

double sqrt() {
  double x_n = 2;
  for (int i=0; i<1000; ++i) {
    x_n = x_n - f(x_n) / df(x_n);
  }

  return x_n;
}
```

1.2 自然数 a の平方根を求める

任意の自然数 a の平方根を求める．この時も同様に，方程式は $f(x) = x^2 - a$ となる.

```
double f(double x, int a) {
  return pow(x, 2) - a;
}

double df(double x, int a) {
  return (f(x+1e-6, a) - f(x, a)) / 1e-6;
}

double sqrt(int a) {
  double x_n = 2;
  for (int i=0; i<1000; ++i) {
    x_n = x_n - f(x_n, a) / df(x_n, a);
  }

  return x_n;
}
```

2　動的計画法

動的計画法は，n 段階決定過程を n 個の 1 段階決定過程の列に直すことによって，多段決定問題を逐次的に解く方法．多段決定問題を動的計画法によって解く手順は，大きく分けて次のステップからなる．

1. 相互に関連を持つ一群の問題からなり，解くべき問題を含む適切な問題群を考える．
2. 最適性の原理に基づき，問題群に含まれる各問題の間の関係を，最適性方程式で表現する．
3. 最適性方程式を解いて最適ポリシーを求める．

2.1　頂上到達の組み合わせ数

ある n ステップからなる階段を登ることを考える．一度の試行で 1 段または 2 段を登ることができるとすると，異なる登り方で頂上に到達する行き方の組み合わせを求める．

```
int climb(int n) {
  if (n <= 2) reutrn n;

  int dp[n];
  dp[0] = 1;
  dp[1] = 2;

  for (int i=2; i<n; ++i) {
    dp[i] = dp[i-1] + dp[i-2];
  }

  return dp[n-1];
}
```

3 連結リスト

3.1 重複要素の削除

ソート済みの連結リストの重複要素を削除することを考える．ここでは 2 つのポインタを活用して一重ループのみで目的を達成することを目指す．用いる二つのポインタ $fast$ と $slow$ は，それぞれ以下のような動きをする．

- fast: リストの先頭から順に調べていき，$slow$ ポインタと値が一致したとき現在指している要素を削除する．
- slow: $fast$ より後方の要素を調べていき，$fast$ と値が一致したときはリストのポインタを繋ぎ直し，値が一致しなかったときは最後に値が一致したときの $fast$ の位置まで動く．

```
struct ListNode {
  int val;
  ListNode *next;
  ListNode(int x) : val(x), next(NULL) { }
};

class Solution {
  public:
    ListNode* deleteDuplicates(ListNode* head) {
      if (head == nullptr) return NULL;

      ListNode* fast = head->next;
      ListNode* slow = head;

      while (slow != nullptr && fast != nullptr) {
        if (slow->val == fast.val) {
          ListNode* tmp = fast;
          fast = fast->next;
          slow->next = fast;
          delete tmp;
        } else {
          fast = fast->next;
          slow = slow->next;
        }
      }

      return head;
    }
}
```