Homework 5: Trial and Error Methods

(Brute Force, Backtracking, Branch and Bound)

I. Lí thuyết phương pháp

Phương pháp vét cạn

Ý tưởng: Phương pháp vét cạn là một phương pháp giải quyết bài toán tối ưu bằng cách duyệt tất cả các khả năng có thể xảy ra để tìm ra giá trị tối ưu của bài toán. Ý tưởng của phương pháp là sử dụng một vòng lặp để duyệt qua tất cả các giá trị có thể của biến tối ưu hóa, tính toán giá trị của hàm mục tiêu tương ứng và lưu trữ giá trị tối ưu nhất đã tìm thấy cho đến hiện tại.

Lược đồ tổng quát của phương pháp vét cạn như sau:

```
BruteForce(D)≡
    for (x1 = d11 .. d1m1)
       for (x2 = d21 .. d2m2)
          .....
            for (xn = dn1 .. dnmn)
               if (f(x1,x2,…xn))
                  <Solution x=(x1,x2,…xn)>;
end.
```

1. Khởi tạo giá trị tối ưu ban đầu.
2. Với mỗi giá trị của biến tối ưu hóa, tính toán giá trị của hàm mục tiêu tương ứng.
3. So sánh giá trị tìm được với giá trị tối ưu hiện tại và cập nhật giá trị tối ưu nếu cần thiết.
4. Lặp lại bước 2 và 3 cho tất cả các giá trị của biến tối ưu hóa.
5. Trả về giá trị tối ưu đã tìm được.

   Tuy nhiên, phương pháp vét cạn có thể trở nên rất tốn kém về mặt thời gian khi số lượng giá trị của biến tối ưu hóa là lớn. Do đó, nó thường được sử dụng trong những trường hợp có số lượng giá trị của biến tối ưu hóa nhỏ.

II. Lập trình

Coding trong foder *./src/bai_2/*

Khớp xâu: Tìm các vị trí xuất hiện của xâu mẫu P trong văn bản T cho trước.

Brute Force algorithm

```
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

Cụ thể, thuật toán Brute Force String Matching có thể được mô tả như sau:

1. Duyệt từng vị trí trong xâu lớn.
2. Với mỗi vị trí, so sánh các ký tự liên tiếp của xâu lớn với các ký tự liên tiếp của xâu con để kiểm tra xem chúng có khớp hay không.
3. Nếu xâu con khớp với một vị trí trong xâu lớn, lưu trữ vị trí đó và tiếp tục duyệt các vị trí tiếp theo.
   Thuật toán Brute Force String Matching có độ phức tạp là O(mn), trong đó m và n lần lượt là độ dài của xâu con và xâu lớn. Độ phức tạp này có thể rất lớn đối với các xâu con và xâu lớn có độ dài lớn.

Dãy nhị phân: Liệt kê các dãy nhị phân có độ dài n.

```
Try(i) ≡
    for (v=0..1)   //v nhận giá trị 0 hoặc 1
        x_i = v;
        if (i= n) printResult (x);
        else Try(i+1) ;
    endfor;
End.                                  Lời gọi ban đầu Try(1)
```

Để liệt kê tất cả các dãy nhị phân có độ dài n bằng phương pháp backtracking, ta có thể sử dụng một mảng boolean để lưu trữ các giá trị 0/1 tại mỗi vị trí trong dãy. Ý tưởng chính của thuật toán backtracking là thử các giá trị có thể cho mỗi vị trí trong mảng, và sau đó đệ quy để tiếp tục xử lý các vị trí tiếp theo.

III. Đặt bài toán, thiết kế, phân tích và triển khai thuật toán

Đề bài toán*: Cho một dãy số nguyên dương gồm n phần tử và một số nguyên dương S, tìm một tập con của dãy số đó có tổng bằng S. Nếu không có tập con nào thỏa mãn, trả về giá trị -1.*

Phân tích bài toán: Đây là bài toán tìm kiếm trong không gian tập con. Ta có thể giải quyết bài toán này bằng phương pháp thử sai Branch and Bound, đây là một kỹ thuật rất hiệu quả để giải quyết các bài toán tối ưu trong không gian tìm kiếm.

Thuật toán Branch and Bound cho phép chúng ta tạo ra một cây tìm kiếm phân nhánh, trong đó mỗi nút đại diện cho một tập con của dãy số, và mỗi cạnh trong cây đại diện cho một phép toán thêm một phần tử vào hoặc loại bỏ một phần tử khỏi tập con. Tại mỗi nút, ta tính tổng các phần tử trong tập con đó, và so sánh với S. Nếu tổng bằng S, ta đã tìm được một tập con thỏa mãn. Nếu không, ta sẽ phân nhánh tiếp theo bằng cách thêm hoặc loại bỏ một phần tử khác để tạo ra các tập con mới.
Để tối ưu hóa thuật toán, chúng ta sử dụng một số phương pháp như chặt tỉa (pruning) để loại bỏ các nhánh không cần thiết. Ví dụ như nếu tổng các phần tử của một tập con đã vượt quá S, ta không cần phân nhánh tiếp và có thể loại bỏ nút đó. Hoặc nếu tổng

các phần tử của một tập con đã bé hơn S, ta cũng không cần phân nhánh tiếp theo vì không thể tạo ra tập con nào có tổng bằng S.
Xây dựng thuật toán:
Đầu vào: một dãy số nguyên dương A gồm n phần tử và một số nguyên dương S. Đầu ra: một tập con của dãy số A có tổng bằng S.

1. Khởi tạo tập con ban đầu là rỗng và đặt mức của nút là 0.
2. Tại mỗi nút i, tính tổng các phần tử trong tập con đó. Nếu tổng bằng S, trả về tập con đó và kết thúc thuật toán.
3. Nếu tổng nhỏ hơn S, phân nhánh tiếp theo bằng cách thêm một phần tử vào tập con. Nếu tổng lớn hơn S, loại bỏ phần tử cuối cùng trong tập con.
4. Áp dụng chặt tỉa (pruning) để loại bỏ các nhánh không cần thiết
   + Nếu tổng các phần tử trong tập con đã vượt quá S, loại bỏ nút đó.
   + Nếu tổng các phần tử của một tập con đã bé hơn S, không phân nhánh tiếp theo vì không thể tạo ra tập con nào có tổng bằng S.
5. Lặp lại bước 2 đến khi tìm được tập con có tổng bằng S hoặc không còn nhánh nào để khám phá.
6. Nếu không tìm thấy tập con nào có tổng bằng S, trả về giá trị -1.

Phân tích thuật toán:

Thuật toán Branch and Bound sẽ tạo ra một cây tìm kiếm phân nhánh với độ sâu không quá n. Tại mỗi nút i của cây, ta thực hiện phép toán thêm hoặc loại bỏ một phần tử khỏi tập con, do đó số lượng nút con của nút i là 2. Các phép toán này sẽ làm cho cây tìm kiếm có chiều rộng tối đa là 2^n, tuy nhiên với các phương pháp chặt tỉa, số lượng nút được giảm xuống đáng kể.

Phương pháp chặt tỉa sẽ giúp ta loại bỏ các nhánh không cần thiết của cây tìm kiếm, giúp thuật toán hoạt động hiệu quả hơn. Nếu ta không sử dụng chặt tỉa, số lượng nút của cây tìm kiếm sẽ là 2^n và thuật toán sẽ rất chậm.

File coding trong ./src/bai_3/