Homework 5

Bài 1:

1. Bài toán đổi tiền
    - File code money.py
2. Bài toán sắp xếp hoạt động sử dụng tài nguyên
    - File code activitySelector.py

Bài 2:

1. Thuật toán Kruskal
    - Ý tưởng:
        i. Sắp xếp các cạnh theo thứ tự tăng dần của trọng số.
        ii. Khởi tạo tập chứa các cạnh của cây khung nhỏ nhất F = ∅
        iii. Xét lần lượt từng cạnh của đồ thị theo thứ tự đã xếp. Nếu cạnh đang xét không tạo ra chu trình với các cạnh trước đó thì ta thêm cạnh này vào F.
        iv. Kết thúc khi tập F có n – 1 phần tử với n là số đỉnh của đồ thị.
    - Lược đồ

```
Kruskal(G) ≡
      // Input: G = (V, E)
      // Output: ET , tập các cạnh của cây bao trùm tối thiểu của G
      E.sort() // sắp xếp các cạnh theo thứ tự trọng số tăng dần
      F = ∅
      Label = [1, 2, … , n ] // gán nhãn cho các đỉnh
      for i in E
            if (F ∪ {i} không tạo ra chu trình)
                  F = F ∪ {i}
      if (length(F) == n – 1) return F
      endf;
End.
```

- Để xét xem cạnh mới khi thêm vào có tạo ra chu trình hay không ta sử dụng kỹ thuật gán nhãn cho các đỉnh.

    i. Khởi tạo nhãn của các đỉnh là chính bằng số thứ tự của các đỉnh
    ii. Nếu hai đầu cạnh e có cùng nhãn (tức là nhãn của e.vertex1 và nhãn của e.vertex2 bằng nhau) thì khi thêm e vào sẽ tạo ra chu trình
    iii. Ngược lại 2 đỉnh của cạnh e có nhãn khác nhau thì ta đưa e vào F và thực hiện công việc ghép nhãn bằng cách:
        - Chọn đỉnh có nhãn nhỏ hơn trong 2 đỉnh, giả sử là V_1
        - Thay thế mọi đỉnh có nhãn bằng với nhãn của đỉnh V_2 bằng nhãn của đỉnh V_1.

2. Chứng minh.
    - Để chứng minh thuật toán tạo ra cây bao trùm tối thiểu của đồ thị, ta sẽ chứng minh kết quả của thuật toán là 1 cây bao trùm(cây khung) và cây đó có tổng trọng số là nhỏ nhất.
        i. Chứng minh cây khung
            - Tại mỗi bước ta đều xét xem các cạnh mới khi thêm vào có tạo ra chu trình với các cạnh đã có hay không, do đó kết quả của thuật toán không tạo ra chu trình.
            - Giả sử F không liên thông, tức là F có thể chứa $k \geq 2$ thành phần mà mỗi thành phần liên thông với nhau. Gọi k thành phần đó là $F_1, F_2, \dots, F_n$. Do F không có chu trình nên F_i cũng không chứa chu trình. Mà F_i liên thông cho nên F_i là cây. Gọi $V(F_i)$ và $E(F_i)$ là số đỉnh và số cạnh của cây F_i => $V(F_i) = E(F_i) + 1$.

              Vậy $V(F) = \sum_{i=1}^{k} V(F_i) = \sum_{i=1}^{k} E(F_i) - k$
              Mà $E(F) = V(F) - 1$ và $k \geq 2$ => mâu thuẫn. Vậy giả sử sai.
            - F liên thông và F không có chu trình => F là cây khung của đồ thị
        ii. Chứng minh nhỏ nhất
            - Giả sử kết quả thuật toán thu được không phải là cây khung nhỏ nhất của đồ thị G. Khi đó tồn tại cây S có tổng các trọng số nhỏ hơn F. kí hiệu là $c(S) < c(F)$.
            - Ký hiệu e_k là 1 cạnh thuộc F mà không thuộc S. Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh e_k sẽ chứa một chu trình C duy nhất đi qua e_k.
            - Ta bỏ 1 cạnh e thuộc C khác cạnh e_k thu được đồ thị mới là S'. Ở mỗi bước của thuật toán ta đều xét cạnh có trọng số nhỏ nhất trước nên $c(e_k) \leq c(e)$ do đó $c(S') \leq c(S)$, đồng thời số cạnh chung của S' và F đã tăng thêm 1 so với số cạnh chung của S và F.
            - Làm tương tự với mọi cạnh thuộc F mà không thuộc S ta có thể biến đổi S thành F và trong mỗi bước tổng độ dài không tăng,

tức là $c(F) \leq c(S)$. Mâu thuẫn thu được chúng tỏ F là cây khung nhỏ nhất.

iii. Vậy kết quả của thuật toán Kruskal cho ta 1 cây khung nhỏ nhất.

3. Độ phức tạp.
    - Ta thấy việc duyệt hết các cạnh thuộc E có độ phức tạp $O(E)$
    - Việc xét xem cạnh mới thêm vào có tạo thành chu trình không dựa vào kĩ thuật dán nhãn có thời gian chạy rất nhỏ $\approx O(1)$
    - Việc sắp xếp các cạnh theo thứ tự tăng dần có độ phức tạp $O(E \log(E))$.
    - Nếu sử dụng cấu trúc heap thì việc lấy ra các cạnh có trọng số nhỏ nhất sẽ mất thời gian hằng số và sắp xếp các cạnh có mất thời gian $O(E \log(E))$
    - Vậy thuật toán có độ phức tạp là $O(E \log(E))$.
4. Thực thi thuật toán
    - File code kruskal.py

Bài 3

1. Bài toán tìm tập có độ dài đơn vị nhỏ nhất
    - Đề bài: Cho tập các số thực S, hãy tìm số lượng nhỏ nhất các khoảng p_i có đọ dài 1 thỏa mãn:
        i. $\bigcap_{i=1}^{n} p_i = \emptyset$
        ii. $X \in \bigcup_{i=1}^{n} p_i \ , \forall x \in s$
    - Ý tưởng thuật toán

        i. Sắp xếp các điểm đã cho trong tập S theo thứ tự tăng dần
        ii. Ta thấy nếu cận trái của khoảng p_i là 1 số thuộc S thì ta có khả năng khoảng đó chứa được nhiều điểm hơn so với các cách chọn khoảng khác, nên số khoảng sẽ ít hơn.
        iii. Áp dụng giải thuật tham lam, ta duyệt các số trong S theo thứ tự từ nhỏ đến lớn. Trong mỗi bước ta lấy số nhỏ nhất của S là p làm cận trái của khoảng, loại bỏ các số thuộc khoảng $[p, p + 1]$ ra khỏi S.
        iv. Làm như vậy cho đến khi S không còn phần tử nào.
    - File code smallestSet.py
    - Lược đồ

```
Partial(S) ≡
      // input tập các số thực S
      // output: các khoảng [p, p + 1] chứa mọi phần tử của S
      S.sort() // sắp xếp các số trong S theo thứ tự tăng dần
      P = ∅
      left = - ∞
      for i in S
            if i > left + 1
                  left = i
                  P = P ∪ {i, i + 1}
                  S \ {i}
            else S \ {i}
      endf
      return P
end
```

- Chứng minh tính đúng
  i. Gọi tập P chứa các khoảng [p, p + 1] là kết quả của thuật toán trên. Giả sử tồn tại tập K chứa các khoảng $[k_i , k_i + 1]$ với số khoảng ít hơn P. Gọi số khoảng của P và K lần lượt là m và n $=>$ m $>$ n
  ii. So sánh lần lượt số điểm nằm trong khoảng $[p_i , p_i + 1]$ và $[k_i , k_i + 1]$. Do mọi điểm của S phải nằm trong các khoảng và m $>$ n nên tồn tại khoảng $[k_i , k_i + 1]$ có số điểm nhiều hơn khoảng $[p_i , p_i + 1]$ tương ứng. Gọi $[p_i , p_i + 1]$ là khoảng đầu tiên có số phần tử ít hơn $[k_i , k_i + 1]$. Do đó $k_i \leq p_i$.
  iii. Gọi x là số thuộc $[k_i , k_i + 1]$ nhưng không thuộc $[p_i , p_i + 1]$
  $=>$ x - $p_i > 1$
  Và x - $k_i < 1$
  Mà $k_i \leq p_i$ $=>$ mâu thuẫn . Giả sử là sai.
  iv. Thuật toán cho ra kết quả là số khoảng nhỏ nhất.
- Đánh giá thuật toán
  i. Nếu tập S đã được sắp xếp theo thứ tự thì thuật toán có độ phức tạp O(n) với n là số phần tử của S.
  ii. Nếu tập S chưa sắp xếp, ta cần sắp xếp lại tập S do đó độ phức tạp của thuật toán là O(n log n)