ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
Ngày 21, Tháng 11, Năm 2024

LÀM BÀI TẬP NHÓM 7

Giáo viên hướng dẫn: ***Nguyễn Thanh Sơn***

Nhóm 2: ***Trần Vinh Khánh***, Nguyễn Thiện Nhân

Mục lục

1. Bài 1

Hướng giải

Bài toán **Set Cover** được giải bằng thuật toán tham lam với các bước sau:

- Ban đầu, tập `universe` chứa tất cả các phần tử cần được bao phủ.
- Tìm tập con S_i bao phủ được nhiều phần tử nhất trong `universe`.
- Thêm tập con S_i vào kết quả và loại bỏ các phần tử đã được bao phủ khỏi `universe`.
- Tiếp tục lặp lại cho đến khi `universe` rỗng.

Thuật toán tham lam có độ phức tạp $O(n \cdot m)$, trong đó n là số phần tử của `universe` và m là số tập con.

Mã C++

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

// Hm tm tp con ti u bng thut ton tham lam
vector<int> setCover(vector<set<int>> &subsets, set<int> universe) {
    vector<int> selectedSubsets; // Cha cc tp con đ c chn

    while (!universe.empty()) {
        // Tm tp con bao ph nhiu phn t nht trong universe
        int bestSubsetIndex = -1;
        int maxCovered = 0;

        for (int i = 0; i < subsets.size(); ++i) {
            int covered = 0;
            for (int elem : subsets[i]) {
                if (universe.count(elem)) {
                    ++covered;
                }
            }
            if (covered > maxCovered) {
                maxCovered = covered;
                bestSubsetIndex = i;
            }
        }

        if (bestSubsetIndex == -1) {
            cerr << "Khng th bao ph tt c cc phn t !" << endl;
            return {};
        }

        // Chn tp con tt nht v cp nht universe
        selectedSubsets.push_back(bestSubsetIndex);
```

```
        for (int elem : subsets[bestSubsetIndex]) {
            universe.erase(elem);
        }
    }

    return selectedSubsets;
}

int main() {
    //  Tp  U
    set<int> universe = {1, 2, 3, 4, 5};

    // Cc  tp  con S
    vector<set<int>> subsets = {
        {1, 2, 3}, // S1
        {2, 4},   // S2
        {3, 4},   // S3
        {4, 5}    // S4
    };

    //  Gii  bi  ton
    vector<int> result = setCover(subsets, universe);

    //  Xut  kt  qu
    if (!result.empty()) {
        cout << " Cn  chn  cc  tp  con sau ( ch  s  bt  đ u  t  0):" << endl;
        for (int idx : result) {
            cout << "S" << idx + 1 << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Đầu ra mẫu

Với đầu vào:

```
universe = {1, 2, 3, 4, 5};
subsets = {{1, 2, 3}, {2, 4}, {3, 4}, {4, 5}};
```

Chương trình sẽ xuất:

```
Cần chọn các tập con sau (chỉ số bắt đầu từ 0):
S1 S4
```

2. Bài 2

1. Thuật toán Tham lam (Greedy)

- Bắt đầu từ một đỉnh bất kỳ.
- Tại mỗi bước, chọn cạnh có trọng số nhỏ nhất nối đến một đỉnh chưa được thăm.

- Lặp lại cho đến khi tất cả các đỉnh được thăm, sau đó quay về đỉnh xuất phát.

Ưu điểm: Dễ cài đặt, hiệu quả cho bài toán nhỏ.
Nhược điểm: Không đảm bảo lời giải tối ưu.

2. Thuật toán 2-Approximation dựa trên cây khung nhỏ nhất (MST)

- Tìm cây khung nhỏ nhất T của đồ thị G bằng thuật toán Prim hoặc Kruskal.
- Duyệt cây T theo thứ tự Euler để tạo chu trình qua tất cả các đỉnh.
- Loại bỏ các đỉnh lặp lại để đảm bảo mỗi đỉnh được thăm đúng một lần.

Ưu điểm: Đảm bảo chi phí không vượt quá 2 lần chi phí tối ưu.
Nhược điểm: Đòi hỏi nhiều bước hơn so với thuật toán Tham lam.

Mã C++

1. Thuật toán Tham lam (Greedy)

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Hm  tm  đ  ng  đi TSP  bng  thut  ton  tham lam
int tspGreedy(vector<vector<int>> &graph, int start) {
    int n = graph.size();
    vector<bool> visited(n, false);
    int current = start;
    visited[current] = true;
    int totalCost = 0;

    for (int step = 0; step < n - 1; ++step) {
        int next = -1;
        int minCost = INT_MAX;

        for (int i = 0; i < n; ++i) {
            if (!visited[i] && graph[current][i] < minCost) {
                minCost = graph[current][i];
                next = i;
            }
        }

        if (next == -1) {
            cerr << "      th  khng  đ y  đ  !" << endl;
            return -1;
        }

        totalCost += minCost;
        visited[next] = true;
        current = next;
    }
```

```
34
35      // Quay  li  đ im   xut   pht
36      totalCost += graph[current][start];
37      return totalCost;
38  }
39
40  int main() {
41      vector<vector<int>> graph = {
42          {0, 10, 15, 20},
43          {10, 0, 35, 25},
44          {15, 35, 0, 30},
45          {20, 25, 30, 0}
46      };
47      int start = 0;
48      cout << "Chi ph TSP (Greedy): " << tspGreedy(graph, start) << endl;
49      return 0;
50  }
```

2. Thuật toán 2-Approximation dựa trên MST

```
1   #include <iostream>
2   #include <vector>
3   #include <set>
4   #include <algorithm>
5   #include <climits>
6   using namespace std;
7
8   struct Edge {
9       int u, v, weight;
10      bool operator<(const Edge &e) const {
11          return weight < e.weight;
12      }
13  };
14
15  // Hm   tm   cy  khung  nh   nht  (MST)  bng  Kruskal
16  vector<Edge> findMST(vector<Edge> &edges, int n) {
17      vector<int> parent(n);
18      for (int i = 0; i < n; ++i) parent[i] = i;
19
20      function<int(int)> find = [&](int u) {
21          return parent[u] == u ? u : parent[u] = find(parent[u]);
22      };
23
24      vector<Edge> mst;
25      sort(edges.begin(), edges.end());
26
27      for (Edge &e : edges) {
28          int uRoot = find(e.u);
29          int vRoot = find(e.v);
30          if (uRoot != vRoot) {
31              mst.push_back(e);
32              parent[uRoot] = vRoot;
33          }
34      }
35
36      return mst;
37  }
```

```cpp

// TSP 2-Approximation
int tspApproximation(vector<vector<int>> &graph) {
    int n = graph.size();
    vector<Edge> edges;

    // Chuyn ma trn trng s thnh danh sch cnh
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            edges.push_back({i, j, graph[i][j]});
        }
    }

    // Tm MST
    vector<Edge> mst = findMST(edges, n);

    // Tnh tng chi ph ca chu trnh gn đ ng
    int totalCost = 0;
    for (Edge &e : mst) {
        totalCost += e.weight;
    }
    totalCost *= 2; // Chu trnh Euler đi qua mi cnh 2 ln

    return totalCost;
}

int main() {
    vector<vector<int>> graph = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };
    cout << "Chi ph TSP (2-Approximation): " << tspApproximation(graph) << endl;
    return 0;
}
```

Kết quả mẫu

- Đầu vào:

$$\text{Graph} = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$$

- Đầu ra:

```
Chi phí TSP (Greedy): 80
Chi phí TSP (2-Approximation): 90
```