Báo cáo phân tích độ phức tạp các giải thuật không đệ quy

Nguyễn Đình Thiên Quang, Đặng Quốc Cường
Giảng viên: Nguyễn Thanh Sơn
Khóa học: CS115

Ngày 8 tháng 10 năm 2024

1 Phân tích giải thuật Huffman Coding

1.1 Bước khởi tạo

Khởi tạo n cây cho mỗi kí tự, do đó bước này có độ phức tạp $O(n)$. Sau đó, thuật toán thực hiện chọn ra 2 cây có chi phí nhỏ nhất, rồi ghép chúng lại với nhau, do đó độ phức tạp của bước này là $O((n-1) \cdot cost(ghp))$. Dễ thấy rằng độ phức tạp của bước sau lớn hơn bước trước nên có thể coi độ phức tạp của thuật toán là $O(n \cdot cost_{computation})$.

1.2 Tối ưu hóa thuật toán

Để tối ưu thuật toán, ta cần tối ưu bước tính $cost_{computation}$, nghĩa là tìm cách chọn ra 2 cây có xác suất nhỏ nhất một cách nhanh chóng. Một cách để làm điều này là dùng priority queue, từ đó giảm bước này xuống cách chọn tối ưu là $O(\log(n))$. Từ đây ta có được thuật toán có độ phức tạp tối ưu là $O(n \log(n))$.

2 Phân tích giải thuật Prim

2.1 Ý tưởng

Để tìm một phiên bản của giải thuật Prim có độ phức tạp $O((n+m)\log(n))$, ta quan tâm tới cách quản lý chặt chẽ các cạnh. Vì độ phức tạp của thuật toán phụ thuộc vào bước tính toán tìm ra cạnh nhỏ nhất chưa được thêm vào cây T (để tạo cây phủ lớn nhất), nên dễ thấy, nếu sử dụng một cấu trúc giống heap (hay priority queue) sẽ tối ưu được lựa chọn này.

2.2 Mã giải thuật Prim

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
#include <functional>
#include <limits.h>

```

```cpp
using namespace std;

typedef pair<int, int> Edge;

int primAlgorithm(int n, vector<vector<Edge>>& adjList) {
    priority_queue<Edge, vector<Edge>, greater<Edge>> pq;
    vector<int> minDist(n, INT_MAX);
    vector<bool> inMST(n, false);
    int mstWeight = 0;

    pq.push({0, 0});
    minDist[0] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        int w = pq.top().first;
        pq.pop();

        if (inMST[u]) continue;

        inMST[u] = true;
        mstWeight += w;

        for (auto& edge : adjList[u]) {
            int v = edge.second;
            int weight = edge.first;

            if (!inMST[v] && weight < minDist[v]) {
                minDist[v] = weight;
                pq.push({weight, v});
            }
        }
    }

    return mstWeight;
}

int main() {
    int n = 5;
    vector<vector<Edge>> adjList(n);

    adjList[0].push_back({2, 1});
    adjList[0].push_back({3, 3});
    adjList[1].push_back({2, 0});
    adjList[1].push_back({1, 2});
    adjList[2].push_back({1, 1});
    adjList[2].push_back({5, 3});
    adjList[2].push_back({4, 4});
    adjList[3].push_back({3, 0});
    adjList[3].push_back({5, 2});
```

```
    adjList[3].push_back({7, 4});
    adjList[4].push_back({4, 2});
    adjList[4].push_back({7, 3});

    int mstWeight = primAlgorithm(n, adjList);
    cout << "Tong trong so cua cay khung nho nhat la: " << mstWeight << e

    return 0;
}
```

3 Phân tích giải thuật Kruskal

3.1 Ý tưởng

Với giải thuật Kruskal, ý tưởng là xếp các cạnh theo độ dài từ nhỏ nhất đến lớn nhất, rồi thực hiện tham lam miễn là các cạnh vẫn tạo thành cây. Do đó, một yếu tố tiên quyết sẽ là tìm ra giải thuật nhanh để kiểm tra chu trình. Ở đây, chúng ta sử dụng cấu trúc dữ liệu Disjoint Set để làm điều này.

3.2 Độ phức tạp

Disjoint Set Union có 2 thao tác, trong đó thao tác hợp nhất có độ phức tạp $O(\log(n))$ nên ta có được giải thuật $O((n+m)\log(n))$.

3.3 Mã giải thuật Kruskal

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Edge {
    int u, v, weight;
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
};

class DSU {
    vector<int> parent, rank;
public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) parent[i] = i;
    }

    int find(int u) {
```

```
        if (u != parent[u]) {
            parent[u] = find(parent[u]);
        }
        return parent[u];
    }

    bool unite(int u, int v) {
        int root_u = find(u);
        int root_v = find(v);

        if (root_u != root_v) {
            if (rank[root_u] < rank[root_v]) {
                swap(root_u, root_v);
            }
            parent[root_v] = root_u;
            if (rank[root_u] == rank[root_v]) {
                rank[root_u]++;
            }
            return true;
        }
        return false;
    }
};

int kruskalAlgorithm(int n, vector<Edge>& edges) {
    sort(edges.begin(), edges.end());

    DSU dsu(n);
    int mstWeight = 0;
    int edgeCount = 0;

    for (auto& edge : edges) {
        if (dsu.unite(edge.u, edge.v)) {
            mstWeight += edge.weight;
            edgeCount++;
            if (edgeCount == n - 1) break;
        }
    }

    return mstWeight;
}

int main() {
    int n = 5;
    vector<Edge> edges = {
        {0, 1, 2},
        {0, 3, 3},
        {1, 2, 1},
        {2, 3, 5},
        {2, 4, 4},
```

```
        {3, 4, 7}
    };

    int mstWeight = kruskalAlgorithm(n, edges);
    cout << "Tong trong so cua cay khung nho nhat la: " << mstWeight << e

    return 0;
}
```