طراحی الگوریتم ها

نیم‌سال دوم ۹۸ـ۹۹

دانشکده‌ی مهندسی کامپیوتر

گردآورندگان: علیرضا دقیق ، نیما فتحی ،مهرداد کرابی ، آراد محمدی

پاسخ نامه تمرین عملی سری سوم — گراف

مسئله‌ی ۱. تو منو امتحان کن

برای حل این سوال ما اینگونه به این قضیه نگاه می‌کنیم که متغیر هارا با راس ها و رابطه شرطی بین آنها را اینگونه تعریف کنیم:

$$Xi - Xj \leqslant Y \longrightarrow (j \to i)$$

ما از راس Xj به Xi یالی با وزن Y می‌کشیم و برای همه شروط این کار را انجام می‌دهیم و به این شکل سوال را به یک گراف مپ می‌کنیم. حال برای حل آن اولا باید چک کنیم که آیا این پرسش دارای جواب درست و قابل قبولی هست یا خیر برای این کار بر روی گراف ایجاد شده الگوریتم Bellman-Ford را اجرا می‌کنیم با V+۱ تکرار و اگر در V+1 امین بار حلقه مسیری آپدیت شد به این معنی است که دور منفی داریم. که خود این قضیه به معنای عدم وجود جواب برای این مجموعه و این دستگاه معادلات است.
اما چر اینگونه است؟
برهان خلف:

$$V_0 \ \to \ V_1 \ - \ V_2 \ \to \ ... \ \to \ V_k \ \to \ V_0$$

وزن منفی دارد:
فرض می‌کنیم دستگاه ما حداقل یک جواب درست دارد:

$$W(V_1) \ - \ W(V_0) \leqslant \ W(V_0, V_1)$$

$$W(V_2) \ - \ W(V_1) \leqslant \ W(V_1, V_2)$$

$$...$$

$$W(V_k) \ - \ W(V_k - 1) \leqslant \ W(V_k - 1, V_k)$$

$$W(V_0) \; - \; W(V_k) \leqslant \; W(V_k, V_0)$$

همه این معادلات را جمع می‌زنیم:

$$0 \; \leqslant \; W(cycle) \; < \; 0$$

تناقض خوردیم پس اگر دوری با وزن منفی داشته باشیم جوابی برای دستگاه معادله وجود ندارد. در حقیقت وقتی یک دور منفی داشته باشیم به این معنی است که هیچ مقداری را برای راس های موجود در این دور نمی‌توان در نظر گرفت که در معادلات یال های آن دور صدق کنند به این معنی که حتما تناقضی با یکی از این معادلات رخ خواهد داد (کمی فکر کنید برایتان بدیهی خواهد بود.)

حال اما قسمت خلاقانه‌تر حل این سوال در صورت نبود دور منفی پیدا کردن یک جواب خواهد بود. بهترین روش در این بخش ایجاد یک تغییر کوچک در گراف اولیه و استفاده از همان Bellman-Ford خواهد بود. برای این کار همواره اولین ایده‌ای که به ذهن می‌رسد و عموما کارگشا است ایده اضافه کردن یک راس با وزن صفر به دیگر راس ها است که اصطلاحا به آن dummy node می‌گویند.
برای این کار از راس جدید به تمامی راس های دیگر یک یال با وزن صفر می‌کشیم این راس و یال های مرتبط با آن هیچ تغییری در اصول اصلی گراف مانند طول مسیرها و دور ها ایجاد نمی‌کند و با توجه به جهت یال‌های اضافه‌شده هیج دور منفی نیز اضافه یا کم نخواهد شد. پس این تغییر اثری بر بخش اول راه حل نخواهد گذاشت و عملا می‌توان همان ابتداکار، دامی نود را اضافه کنیم.اضافه کردن این بال ها به این معنا است که حالا حداقل یک مسیر از این راس (s) به تمامی راس های دیگر با طول شمارا وجود خواهد شد.
حال برای پیدا کردن یک جواب قانونی و مورد قبول کافی است از راس اضافه شده الگوریتم Bellman-Ford را بزنیم و صرفا یکبار اضافه حلقه را پیمایش کنیم تا وجود دور منفی را چک کنیم. مقادیری که برای کوتاهترین فاصله از هر راس بدست می‌اید یک جواب مورد قبول خواهد بود.
چرا؟ از روی قضیه نامساوی مثلث اثبات خواهد شد:

$$d(s,u) + W(u,u) \geqslant d(s,v) \leftrightarrow d(s,v) - d(s,u) \leqslant W(u,v) \leftrightarrow Xv - Xu \leqslant W(u,v)$$

$$X_v = d(S,v)$$

d(s،v): کوتاهترین مسیر از s به v
برای درک بهتر ای پرسش و راه حل آن عبارت system of difference constraint را سرچ کنید.

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

struct Edge {
    int src, dest, weight;
};

struct Graph {
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[graph->E];
    return graph;
}

void isNegCycleBellmanFord(struct Graph* graph,
                           int src)
{
    int V = graph->V;
    int E = graph->E;
    int* dist = new  int[V];
    int* arr = new int[1];
    arr[0] = -1000000;

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }


```

```
    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            cout << "NO";
            return;

        }
    }
    cout<<"YES"<<endl;
    for (int i = 1; i <V ; i++) {
        cout << dist[i]<<" ";

    }
}

int main()
{
    int V , E;

    cin >> V >> E;

    struct Graph* graph = createGraph(V+1, E+V+1);

    for (int i = 0 ; i<E ; i++){
        int a , b , c;
        cin >> a >>b>>c;
        graph->edge[i].src = b;
        graph->edge[i].dest = a;
        graph->edge[i].weight = c;
    }
    for (int i = 0 ; i < V+1 ; i++){
        graph->edge[E+i].src = 0;
        graph->edge[E+i].dest = i;
        graph->edge[E+i].weight = 0;
    }
   isNegCycleBellmanFord(graph,0);


    return 0;
}
```

مسئله‌ی ۲. شاه شهردار

مساله ما در واقع همان مساله k-minimum-spanning tree می باشد با این تفاوت که باید یک راس خاص حتما در مجموعه وجود داشته باشد. برای حل این سوال چیزی شبیه به الگوریتم

kruskal روی گراف اجرا میکنیم. به این صورت که ابتدا دودسته راس داریم .یکی آنهایی که در درخت کمینه ما هستند که در ابتدا فقط راس مرکز شامل این حالت میشود و دیگری آنهایی که متعلق به این مجموعه نیستند. حال مشابه با الگوریتم kruskal راس های جدید را به این مجموعه اضافه میکنیم

```
#include <bits/stdc++.h>
using namespace std;




struct Edge {
    int src, dest, weight;
};

struct Graph {
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[graph->E];
    return graph;
}



int minKey(int key[], bool mstSet[],int V)
{
    int min = INT_MAX, min_index;

    for (int i = 0; i < V; i++)
        if (mstSet[i] == false && key[i] < min)
            min = key[i],
                    min_index = i;

    return min_index;
}


int primMST(struct Graph* g,int srcs, int n ) {
    int V, E = 0;
```

```
    V = g->V;
    E = g->E;
    int parent[V];

    int key[V];

    bool mstSet[V];

    int maxnum = -1;
    int k = 0;


    int graph[V][V];
    for (int j = 0; j <V ; ++j) {
        for (int i = 0; i < V; ++i) {
            graph[j][i]=0;

        }

    }
    for (int i = 0; i < E; i++) {
        int a = g->edge[i].src;
        int b = g->edge[i].dest;
        int c = g->edge[i].weight;
        graph[a][b] = c;
    }


    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX, mstSet[i] = false;
    }

    key[srcs] = 0;
    parent[srcs] = srcs; // First node is always root of MST

    for (int count = 0; count < V ; count++)
    {
        int u = minKey(key, mstSet, V);

        int temp = parent[u];

        mstSet[u] = true;

        k ++;


        if (graph[u][temp] > maxnum)
            maxnum = graph[u][temp];
```

```
        if (k == n)
            break;


        for (int v = 0; v < V; v++) {

            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]
                parent[v] = u, key[v] = graph[u][v];

        }
    }

    int c = 0;
    for (int i = 0 ;i < V; i ++){
        if (key[i] !=INT_MAX)
            c++;
    }
    if (c < n)
        return - 1;

    return maxnum;

}

// Driver code
int main()
{
    int V,E;
    std::cin >> V >> E;
    struct Graph* graph = createGraph(V, 2*E);

    for (int i = 0 ; i < E; i++){
        int a,b,c;
        std::cin >> a >> b >> c;
        graph->edge[i].src=a-1;
        graph->edge[i].dest=b-1;
        graph->edge[i].weight=c;
        graph->edge[E+i].src=b-1;
        graph->edge[E+i].dest=a-1;
        graph->edge[E+i].weight=c;
    }

    int srcs, k;
    std::cin >> srcs >> k;
    // Print the solution
    int ans = 0;
    ans = primMST(graph, srcs-1, k);
    std::cout << ans;
```

```
    return 0;
}
```

مسئله‌ی ۳. بازی عجیب

یکی از روش های ساده برای حل این سوال به صورت زیر است.ابتدا یک گراف می سازیم که در هر راس آن نشان می دهیم که در کدام خانه شطرنج قرار داریم ، کدوم وجه تاس در زیر قرار گرفته و جهت چرخش تاس چجوریه.که در مجموع ۶۴ * ۶ * ۴ تا راس در این گراف قرار می گیره .هر راس با ۴ تا راس دیگر حداکثر همسایگی داره.حالا اگه از راس گفته شده شروع کنی ، راس نهایی یکی از ۲۴ تا راسیه که با چرخش های مختلف تاس به دست میاد.پس کافیه که ۲۴ تا کروسکال بزنیم.راه حل کمی پیچیده تر هم این است که کل گراف رو کامل نسازیم و در هر مرحله قسمت های مورد نیاز رو بسازیم.همینطور میشه که به جای ۲۴ تا کروسکال ، یدونه بزنیم.کدش هم به صورت زیر میشه.

```
#include <iostream>
#include <climits>
using namespace std;
#define N 8
#define V N*N*6*4


int minDistance(const int dist[], const bool sptSet[])
{
    int min = INT_MAX, min_index = 0;

    for (int v = 0; v < V; v++) {
        if (!sptSet[v] &&
            dist[v] <= min)
            min = dist[v], min_index = v;
    }

    if (min == INT_MAX)
        return -1;
    return min_index;
}

void setPath(const int parent[], int j, int* res)
{
    if (parent[j] == -1)
        return;

    setPath(parent, parent[j], res);

    res[0]++;
    res[res[0]+1] = j;
```

```
}

int setSolution(int parent[], const int dist[], int dest, int* res)
{
    res[1] = dist[dest];
    setPath(parent, dest, res);
    return 0;
}

void dijkstra(int **graph, int src, int dest, int* res)
{
    int dist[V];

    bool sptSet[V];

    int parent[V];

    for (int i = 0; i < V; i++)
    {
        parent[i] = -1;
        dist[i] = INT_MAX;
        sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++)
    {
        int u = minDistance(dist, sptSet);

        if (u == -1) {
            res[1] = INT_MAX;
            return;
        }

        if (u == dest)
            break;

        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] >= 0 &&
                dist[u] + graph[u][v] < dist[v])
            {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }
```

```

    setSolution(parent, dist, dest, res);
}

bool hasRightBlock(int currentBlock){
    return (currentBlock + 1) % N != 0;
}

bool hasLeftBlock(int currentBlock){
    return currentBlock % N != 0;
}

bool hasUpBlock(int currentBlock){
    return currentBlock <= 55;
}

bool hasBottomBlock(int currentBlock){
    return currentBlock >= N;
}

int getRightBlock(int currentBlock){
    return currentBlock + 1;
}

int getLeftBlock(int currentBlock){
    return currentBlock - 1;
}

int getUpBlock(int currentBlock){
    return currentBlock + N;
}

int getBottomBlock(int currentBlock){
    return currentBlock - N;
}

int index_of__(const int arr[], int length, int value){
    for (int i = 0; i < length; ++i) {
        if (arr[i] == value)
            return i;
    }
    return -1;
}

pair<int, int> getTurnLeftCube(int currentState, int currentTurn,
                               int** neighbors){
    int frontState = neighbors[currentState][currentTurn];
    int leftState = neighbors[currentState]
```

```
    [(index_of__(neighbors[currentState], 4, frontState) + 3) % 4];
    return make_pair(leftState,
    index_of__(neighbors[leftState], 4, frontState));
}

 pair<int, int> getTurnRightCube(int currentState, int currentTurn,
                                 int** neighbors){
    int frontState = neighbors[currentState][currentTurn];
    int rightState = neighbors[currentState]
    [(index_of__(neighbors[currentState], 4, frontState) + 1) % 4];
    return  make_pair(rightState,
    index_of__(neighbors[rightState], 4, frontState));
}

 pair<int, int> getTurnUpCube(int currentState, int currentTurn,
                              int** neighbors){
    int frontState = neighbors[currentState][currentTurn];
    return  make_pair(5-frontState,
    index_of__(neighbors[5-frontState], 4, currentState));
}

 pair<int, int> getTurnBottomCube(int currentState, int currentTurn,
                                  int** neighbors){
    int frontState = neighbors[currentState][currentTurn];
    return  make_pair(frontState,
    index_of__(neighbors[frontState], 4, 5-currentState));
}

int getCost(int side, int near, int far, int top,
                        int right, int bottom, int left){
    if (side == 0)
        return bottom;
    if (side == 1)
        return near;
    if (side == 2)
        return left;
    if (side == 3)
        return right;
    if (side == 4)
        return far;
    if (side == 5)
        return top;
    return 0;
}

int main()
{
    int** neighbors = new int*[6];
```

```

    neighbors[0] = new int[4]{1, 3, 4, 2};
    neighbors[1] = new int[4]{0, 2, 5, 3};
    neighbors[2] = new int[4]{0, 4, 5, 1};
    neighbors[3] = new int[4]{0, 1, 5, 4};
    neighbors[4] = new int[4]{0, 3, 5, 2};
    neighbors[5] = new int[4]{1, 2, 4, 3};

     string start;
     string end;
    int near;
    int far;
    int top;
    int right;
    int bottom;
    int left;
     cin >> start;
     cin >> end;
     cin >> near; //1
     cin >> far; //4
     cin >> top; //5
     cin >> right; //3
     cin >> bottom; //0
     cin >> left; //2

    int** graph = new int*[V];
    for(int i = 0; i < V; ++i)
        graph[i] = new int[V];

    for (int j = 0; j < V; ++j) {
        for (int i = 0; i < V; ++i) {
            graph[i][j] = -1;
        }
    }

    for (int i = 0; i < V; ++i) {
        int block = i / 24;
        int side = (i / 4) % 6;
        int turn = i % 4;

        if (hasUpBlock(block)) {
            pair<int, int> res = getTurnUpCube(side, turn, neighbors);
            graph[i][getUpBlock(block) * 24 +
                            res.first * 4 + res.second]
            =getCost(res.first, near, far, top, right, bottom, left);
        }

        if (hasBottomBlock(block)) {
```

```
            pair<int, int> res = getTurnBottomCube(side, turn, neighbors)
            graph[i][getBottomBlock(block) * 24 +
                            res.first * 4 + res.second]
            =getCost(res.first, near, far, top, right, bottom, left);
        }

        if (hasLeftBlock(block)) {
            pair<int, int> res = getTurnLeftCube(side, turn, neighbors);
            graph[i][getLeftBlock(block) * 24 +
                            res.first * 4 + res.second]
            =getCost(res.first, near, far, top, right, bottom, left);
        }

        if (hasRightBlock(block)) {
            pair<int, int> res = getTurnRightCube(side, turn, neighbors);
            graph[i][getRightBlock(block) * 24 +
                             res.first * 4 + res.second]
            =getCost(res.first, near, far, top, right, bottom, left);
        }
    }


    int startBlock = ((int) start.at(0) - 97) + N * (((int) start.at(1))
    int endBlock = ((int) end.at(0) - 97) + N * (((int) end.at(1)) - 49);

    int minVal = INT_MAX;
    int minRes[100];
    for (int l = 0; l < 24; ++l) {
        int res[100];
        res[0] = 0;
        dijkstra(graph, startBlock * 24 , endBlock * 24 + l, res);
        if (res[1] <= minVal) {
            minVal = res[1];
             copy( begin(res),  end(res),  begin(minRes));
        }
    }


     cout << minRes[1] + bottom << " " <<
     (char)(startBlock % 8 + 97) << startBlock / 8 + 1 << " ";
    for (int k = 2; k <= minRes[0]+1; ++k) {
         cout << (char)((minRes[k] / 24) % 8 + 97)
         << minRes[k] / 192 + 1 << " ";
    }
     cout <<  endl;

    return 0;
}
```

موفق باشید :)