

The Azinaa Code Notebook

December 10, 2017

Contents

0.1	Graph	3
0.1.1	BFS	3
0.1.2	Bellman-Ford	3
0.1.3	Topological-Sorting	4
0.1.4	Dijkstra	5
0.1.5	Floyd Warshall	6
0.1.6	Strongly Connected Component	7
0.1.7	Minimum Spanning Tree	9
0.1.8	Maximum-BPM	10
0.2	Flow	11
0.2.1	Max-Flow	11
0.2.2	Min-Cut	12
0.3	Geometry	14
0.3.1	Convex-Hull	14
0.3.2	Shoelace Formula (python)	16
0.4	String	16
0.4.1	LCS	16
0.4.2	LIS	17

0.4.3	KMP	18
0.5	Other	19
0.5.1	Disjoint Set	19
0.5.2	Date Transformation	20
0.5.3	make CIN and COUT Run Faseter	21

0.1 Graph

0.1.1 BFS

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int SIZE = 100; //Vertex size
4 int main(){
5     int source = 0;
6     vector<int> Graph[SIZE];
7     bool visited[SIZE];
8     int distance[SIZE];
9     fill(visited, visited+SIZE, 0);
10    fill(distance, distance+SIZE, INT_MAX);
11    queue<int> q;
12    q.push(source);
13    visited[source] = true;
14    distance[source] = 0;
15    while(!q.empty()){
16        int u = q.front();
17        q.pop();
18        for(int i = 0 ; i < Graph[u].size(); i++){
19            if(visited[Graph[u][i]]){
20                continue;
21            }
22            visited[Graph[u][i]] = true;
23            distance[Graph[u][i]] = distance[u] + 1;
24            q.push(Graph[u][i]);
25    }
```

```
26     }
27     return 0;
28 }
```

0.1.2 Bellman-Ford

```
1 // A C++ program for Bellman-Ford's single source
2 #include <bits/stdc++.h>
3
4 struct Edge
5 {
6     int src, dest, weight;
7 };
8
9 struct Graph
10 {
11     int V, E; //Size Of Graph
12     struct Edge* edge;
13 };
14
15 struct Graph* createGraph(int V, int E)
16 {
17     struct Graph* graph = new Graph;
18     graph->V = V;
19     graph->E = E;
20     graph->edge = new Edge[E];
21     return graph;
22 }
23
```

```

24 void BellmanFord(struct Graph* graph, int src, int*
    dist)
25 {
26     int V = graph->V;
27     int E = graph->E;
28
29     for (int i = 0; i < V; i++)
30         dist[i] = INT_MAX;
31     dist[src] = 0;
32
33     for (int i = 1; i <= V-1; i++)
34     {
35         for (int j = 0; j < E; j++)
36         {
37             int u = graph->edge[j].src;
38             int v = graph->edge[j].dest;
39             int weight = graph->edge[j].weight;
40             if (dist[u] != INT_MAX && dist[u] +
                weight < dist[v])
41                 dist[v] = dist[u] + weight;
42         }
43     }
44
45     for (int i = 0; i < E; i++)
46     {
47         int u = graph->edge[i].src;
48         int v = graph->edge[i].dest;
49         int weight = graph->edge[i].weight;
50         if (dist[u] != INT_MAX && dist[u] + weight <
            dist[v])
51             printf("Graph contains negative weight
                cycle"); //Hint!
52     }
53
54     //printArr(dist, V); //Finishing Algo
55
56     return;
57 }
58
59 int main()
60 {
61     int V = 5; // Number of vertices in graph
62     int E = 8; // Number of edges in graph
63     struct Graph* graph = createGraph(V, E);
64     graph->edge[0].src = 0;
65     graph->edge[0].dest = 1;
66     graph->edge[0].weight = -1;
67     int dist[5];
68     BellmanFord(graph, 0, dist);
69     return 0;
70 }

```

0.1.3 Topological-Sorting

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int SIZE = 100; //Graph Size Vertex
4 int visited[SIZE];

```

```

5 vector <int> Graph[SIZE];
6 stack<int> ans;
7 int toposort(int u)
8 {
9     if(visited[u] == 1)
10     {
11         return -1; //cycle
12     }
13     if(visited[u] == 2)
14     {
15         return 0;
16     }
17     visited[u] = 1;
18     for(int i = 0 ; i < Graph[u].size() ; i++)
19     {
20         toposort(Graph[u][i]);
21     }
22     ans.push(u);
23     return 0;
24 }
25
26 int main(){
27     return 0;
28 }

```

0.1.4 Dijkstra

```

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

```

```

3 #define endl '\n'
4 #define pii pair<int,int>
5 #define F first
6 #define S second
7 #define mp make_pair
8 #define pb emplace_back
9
10 bool vis[100001];
11 int dis[100001];
12 vector<pii> a[100001];
13
14 class prioritize {
15 public: bool operator()(pii &p1 , pii &p2) {
16     return p1.S > p2.S;
17 }
18 };
19
20 int Dijkstra(int s, int n) {
21     for (int i = 0; i <= n; i++) {
22         vis[i] = false;
23         dis[i] = INT_MAX;
24     }
25     priority_queue<pii, vector<pii>, prioritize> pq;
26     pq.push(mp(s, dis[s] = 0));
27     while (!pq.empty()) {
28         pii cur = pq.top(); pq.pop();
29         int cv = cur.F, cw = cur.S;
30         if (vis[cv]) continue;
31         vis[cv] = true;

```

```

32     for (pii x : a[cv]) {
33         if (!vis[x.F] && (cw + x.S) < dis[x.F]) {
34             pq.push(mp(x.F, dis[x.F] = cw + x.S));
35         }
36     }
37 }
38 }
39
40 int main() {
41     int tc;
42     cin >> tc;
43     while (tc--) {
44         int v1, v2, w, n, m;
45         cin >> n >> m;
46         for (int i = 0; i <= n; i++) {
47             a[i].clear();
48         }
49         for (int i = 0; i < m; i++) {
50             cin >> v1 >> v2 >> w;
51             a[v1].pb(mp(v2, w));
52         }
53         int s;
54         cin >> s;
55         Dijkstra(s, n);
56         for (int i = 1; i <= n; i++) {
57             if (dis[i] != INT_MAX) {
58                 cout << dis[i] << "␣";
59             } else {
60                 cout << "-1␣";

```

```

61     }
62 }
63 }
64 return 0;
65 }

```

0.1.5 Floyd Warshall

```

1  // C Program for Floyd Warshall Algorithm
2  #include<stdio.h>
3
4  #define V 4
5  #define INF 99999999
6
7  void floydWarshall (int graph[][V])
8  {
9      int dist[V][V], i, j, k;
10     for (i = 0; i < V; i++)
11         for (j = 0; j < V; j++)
12             dist[i][j] = graph[i][j];
13     for (k = 0; k < V; k++)
14     {
15         for (i = 0; i < V; i++)
16         {
17             for (j = 0; j < V; j++)
18             {
19                 if (dist[i][k] + dist[k][j] < dist[i][j])
20                     dist[i][j] = dist[i][k] + dist[k][j];
21             }

```

```

22     }
23 }
24 // All distances -> dist
25 }
26
27 int main()
28 {
29     /* example Graph:
30         10
31         (0)----->(3)
32         |           /\
33         5 |         |
34         |         | 1
35         \\/         |
36         (1)----->(2)
37             3          */
38     int graph[V][V] =
39     {
40         {0, 5, INF, 10},
41         {INF, 0, 3, INF},
42         {INF, INF, 0, 1},
43         {INF, INF, INF, 0}
44     };
45
46     // Print the solution
47     floydWarshall(graph);
48     return 0;
49 }

```

0.1.6 Strongly Connected Component

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int SIZE = 50001;
4 bool visited[SIZE];
5 //Input: vector<int> Graph, vector<int> Reverse Graph
6 // int Number Of vertexes
7 //Output:
8 //Strongly connected componnet Graph->
9 //---->SCC_Graph
10 //List of Each Component -->
11 //----->ListOfEachSCC
12 //to see each node blongs to where
13 //-----SCC_list
14 //SCC returns the size of SCC graph
15 vector<int> SCC_Graph[SIZE];
16 vector<int> ListOfEachSCC[SIZE];
17 int SCC_List[SIZE];
18
19 void TopolSort(vector<int> g[],int u, stack<int>& ans)
20 {
21     if(visited[u]){
22         return;
23     }
24     visited[u] = true;
25     for(int i = 0 ; i < g[u].size() ; i++){
26         TopolSort(g,g[u][i],ans);

```



```

26     }
27     ans.push(u);
28 }
29
30 void DFSR(vector<int> g[],int u, int counter){
31     if(visited[u]){
32         return;
33     }
34     visited[u] = true;
35     for(int i = 0 ; i < g[u].size() ; i++){
36         DFSR(g,g[u][i], counter);
37     }
38     ListOfEachSCC[counter].push_back(u);
39     SCC_List[u] = counter;
40 }
41
42 int SCC(vector<int> graph[], vector<int>
graph_reverce[],int v){
43     fill(visited, visited+v, false);
44     for(int i = 0 ; i < v ; i++){
45         SCC_Graph[i].clear();
46         ListOfEachSCC[i].clear();
47         SCC_List[i] = 0;
48     }
49     stack<int> TopolSorted;
50     for(int i = 0 ; i < v ; i++){
51         if(visited[i]){
52             continue;
53         }

```

```

54         TopolSort(graph,i, TopolSorted);
55     }
56     fill(visited, visited+v, false);
57     fill(SCC_List, SCC_List+v, 0);
58     int counter = 0;
59     while(!TopolSorted.empty()){
60         int u = TopolSorted.top();
61         TopolSorted.pop();
62         if(visited[u]){
63             continue;
64         }
65         DFSR(graph_reverce, u, counter);
66         counter++;
67     }
68     for(int i = 0 ; i < counter ; i++){
69         for(int j = 0 ; j < ListOfEachSCC[i].size() ;
j++){
70             int u = ListOfEachSCC[i][j];
71             for(int k = 0 ; k < graph[u].size() ; k
++){
72                 int w = graph[u][k];
73                 if(SCC_List[u] != SCC_List[w]){
74                     SCC_Graph[SCC_List[u]].push_back(
SCC_List[w]);
75                 }
76             }
77         }
78     }
79     return counter;

```

80 }

0.1.7 Minimum Spanning Tree

```

1  //C++ program for Prim's Minimum Spanning Tree (MST)
   algorithm.
2  #include <stdio.h>
3  #include <limits.h>
4
5  #define V 5 //Graph Size
6
7  int minKey(int key[], bool mstSet[])
8  {
9      int min = INT_MAX, min_index;
10     for (int v = 0; v < V; v++)
11         if (mstSet[v] == false && key[v] < min)
12             min = key[v], min_index = v;
13     return min_index;
14 }
15
16 int printMST(int parent[], int n, int graph[V][V])
17 {
18     printf("Edge\t\tWeight\n");
19     for (int i = 1; i < V; i++)
20         printf("%d\t-%d\t\t\t\t%d\n", parent[i], i, graph[
21             i][parent[i]]);
22 }
23 void primMST(int graph[V][V])

```

```

24 {
25     int parent[V];
26     int key[V];
27     bool mstSet[V];
28     for (int i = 0; i < V; i++)
29         key[i] = INT_MAX, mstSet[i] = false;
30     key[0] = 0;
31     parent[0] = -1;
32     for (int count = 0; count < V-1; count++)
33     {
34         int u = minKey(key, mstSet);
35         mstSet[u] = true;
36         for (int v = 0; v < V; v++)
37             if (graph[u][v] && mstSet[v] == false &&
38                 graph[u][v] < key[v])
39                 parent[v] = u, key[v] = graph[u][v];
40     }
41     printMST(parent, V, graph); //print Solution
42 }
43 int main()
44 {
45     /* Let us create the following graph
46         2      3
47         (0) -- (1) -- (2)
48         /   / \   /
49        6/ 8/  \5 /7
50         / /    \ /
51        (3) ----- (4)

```

```

52         9          */
53     int graph[V][V] =
54         {{0, 2, 0, 6, 0},
55          {2, 0, 3, 8, 5},
56          {0, 3, 0, 0, 7},
57          {6, 8, 0, 0, 9},
58          {0, 5, 7, 9, 0},
59          };
60     primMST(graph);
61     return 0;
62 }

```

0.1.8 Maximum-BPM

```

1  // A C++ program to find maximal Bipartite matching.
2  #include <iostream>
3  #include <string.h>
4  using namespace std;
5
6  #define M 6 // Size Of Graph M*N
7  #define N 6
8
9  bool bpm(bool bpGraph[M][N], int u, bool seen[], int
    matchR[])
10 {
11     for (int v = 0; v < N; v++)
12     {
13         if (bpGraph[u][v] && !seen[v])
14             {

```

```

15         seen[v] = true; // Mark v as visited
16         if (matchR[v] < 0 || bpm(bpGraph, matchR[
            v], seen, matchR))
17             {
18                 matchR[v] = u;
19                 return true;
20             }
21     }
22 }
23 return false;
24 }
25
26 int maxBPM(bool bpGraph[M][N])
27 {
28     int matchR[N]; //List Of Matches
29
30     memset(matchR, -1, sizeof(matchR));
31
32     int result = 0;
33     for (int u = 0; u < M; u++)
34     {
35         bool seen[N];
36         memset(seen, 0, sizeof(seen));
37         if (bpm(bpGraph, u, seen, matchR))
38             result++;
39     }
40     return result;
41 }
42

```

```

43 int main()
44 {
45     bool bpGraph[M][N] = { {0, 1, 1, 0, 0, 0},
46                             {1, 0, 0, 1, 0, 0},
47                             {0, 0, 1, 0, 0, 0},
48                             {0, 0, 1, 1, 0, 0},
49                             {0, 0, 0, 0, 0, 0},
50                             {0, 0, 0, 0, 0, 1}
51     };
52
53     cout << "Maximum Matching" << maxBPM(bpGraph);
54
55     return 0;
56 }

```

0.2 Flow

0.2.1 Max-Flow

```

1  // C++ program for implementation of Ford Fulkerson
   algorithm
2  #include <iostream>
3  #include <limits.h>
4  #include <string.h>
5  #include <queue>
6  using namespace std;
7
8  #define V 6
9

```

```

10 bool bfs(int rGraph[V][V], int s, int t, int parent
    [])
11 {
12     bool visited[V];
13     memset(visited, 0, sizeof(visited));
14     queue <int> q;
15     q.push(s);
16     visited[s] = true;
17     parent[s] = -1;
18     while (!q.empty())
19     {
20         int u = q.front();
21         q.pop();
22
23         for (int v=0; v<V; v++)
24         {
25             if (visited[v]==false && rGraph[u][v] >
                0)
26             {
27                 q.push(v);
28                 parent[v] = u;
29                 visited[v] = true;
30             }
31         }
32     }
33     return (visited[t] == true);
34 }
35
36 int fordFulkerson(int graph[V][V], int s, int t)

```

```

37 {
38     int u, v;
39     int rGraph[V][V];
40     for (u = 0; u < V; u++)
41         for (v = 0; v < V; v++)
42             rGraph[u][v] = graph[u][v];
43     int parent[V];
44     int max_flow = 0;
45     while (bfs(rGraph, s, t, parent))
46     {
47         int path_flow = INT_MAX;
48         for (v=t; v!=s; v=parent[v])
49         {
50             u = parent[v];
51             path_flow = min(path_flow, rGraph[u][v]);
52         }
53
54         for (v=t; v != s; v=parent[v])
55         {
56             u = parent[v];
57             rGraph[u][v] -= path_flow;
58             rGraph[v][u] += path_flow;
59         }
60         max_flow += path_flow;
61     }
62     return max_flow;
63 }
64
65 int main()

```

```

66 {
67     int graph[V][V] = { {0, 16, 13, 0, 0, 0},
68                          {0, 0, 10, 12, 0, 0},
69                          {0, 4, 0, 0, 14, 0},
70                          {0, 0, 9, 0, 0, 20},
71                          {0, 0, 0, 7, 0, 4},
72                          {0, 0, 0, 0, 0, 0}
73     };
74     cout << "The maximum possible flow is"
75           << fordFulkerson(graph, 0, 5);
76     return 0;
77 }

```

0.2.2 Min-Cut

```

1  // C++ program for finding minimum cut using Ford-
   Fulkerson
2  #include <iostream>
3  #include <limits.h>
4  #include <string.h>
5  #include <queue>
6  using namespace std;
7
8  // Number of vertices in given graph
9  #define V 6
10
11 int bfs(int rGraph[V][V], int s, int t, int parent[])
12 {
13     bool visited[V];

```

```

14     memset(visited, 0, sizeof(visited));
15
16     queue <int> q;
17     q.push(s);
18     visited[s] = true;
19     parent[s] = -1;
20
21     while (!q.empty())
22     {
23         int u = q.front();
24         q.pop();
25
26         for (int v=0; v<V; v++)
27         {
28             if (visited[v]==false && rGraph[u][v] >
29                 0)
30             {
31                 q.push(v);
32                 parent[v] = u;
33                 visited[v] = true;
34             }
35         }
36         return (visited[t] == true);
37     }
38
39 void dfs(int rGraph[V][V], int s, bool visited[])
40 {
41     visited[s] = true;

```

```

42     for (int i = 0; i < V; i++)
43         if (rGraph[s][i] && !visited[i])
44             dfs(rGraph, i, visited);
45 }
46
47 void minCut(int graph[V][V], int s, int t)
48 {
49     int u, v;
50     int rGraph[V][V];
51     for (u = 0; u < V; u++)
52         for (v = 0; v < V; v++)
53             rGraph[u][v] = graph[u][v];
54     int parent[V];
55     while (bfs(rGraph, s, t, parent))
56     {
57         int path_flow = INT_MAX;
58         for (v=t; v!=s; v=parent[v])
59         {
60             u = parent[v];
61             path_flow = min(path_flow, rGraph[u][v]);
62         }
63         for (v=t; v != s; v=parent[v])
64         {
65             u = parent[v];
66             rGraph[u][v] -= path_flow;
67             rGraph[v][u] += path_flow;
68         }
69     }
70     //Finishing...

```

```

71     bool visited[V];
72     memset(visited, false, sizeof(visited));
73     dfs(rGraph, s, visited);
74     for (int i = 0; i < V; i++)
75         for (int j = 0; j < V; j++)
76             if (visited[i] && !visited[j] && graph[i][j]
77                 )
78                 cout << i << "□-□" << j << endl;
79     return;
80 }
81
82 int main()
83 {
84     int graph[V][V] = { {0, 16, 13, 0, 0, 0},
85                         {0, 0, 10, 12, 0, 0},
86                         {0, 4, 0, 0, 14, 0},
87                         {0, 0, 9, 0, 0, 20},
88                         {0, 0, 0, 7, 0, 4},
89                         {0, 0, 0, 0, 0, 0}
90                     };
91
92     minCut(graph, 0, 5);
93
94     return 0;
95 }

```

0.3 Geometry

0.3.1 Convex-Hull

```

1  // A C++ program to find convex hull of a set of
2  points. Refer
3  #include <iostream>
4  #include <stack>
5  #include <stdlib.h>
6  using namespace std;
7
8  struct Point
9  {
10     int x, y;
11 };
12 Point p0;
13
14 Point nextToTop(stack<Point> &S)
15 {
16     Point p = S.top();
17     S.pop();
18     Point res = S.top();
19     S.push(p);
20     return res;
21 }
22
23 int swap(Point &p1, Point &p2)
24 {

```

```

25     Point temp = p1;
26     p1 = p2;
27     p2 = temp;
28 }
29
30 int distSq(Point p1, Point p2)
31 {
32     return (p1.x - p2.x)*(p1.x - p2.x) +
33           (p1.y - p2.y)*(p1.y - p2.y);
34 }
35
36 int orientation(Point p, Point q, Point r)
37 {
38     int val = (q.y - p.y) * (r.x - q.x)
39             - (q.x - p.x) * (r.y - q.y);
40     if (val == 0) return 0;
41     return (val > 0)? 1: 2;
42 }
43
44 int compare(const void *vp1, const void *vp2)
45 {
46     Point *p1 = (Point *)vp1;
47     Point *p2 = (Point *)vp2;
48
49     int o = orientation(p0, *p1, *p2);
50     if (o == 0)
51         return (distSq(p0, *p2) >= distSq(p0, *p1))?
52             -1 : 1;
53

```

```

54     return (o == 2)? -1: 1;
55 }
56
57 void convexHull(Point points[], int n)
58 {
59     int ymin = points[0].y, min = 0;
60     for (int i = 1; i < n; i++)
61     {
62         int y = points[i].y;
63         if ((y < ymin) || (ymin == y &&
64             points[i].x < points[min].x))
65             ymin = points[i].y, min = i;
66     }
67
68     swap(points[0], points[min]);
69
70     p0 = points[0];
71     qsort(&points[1], n-1, sizeof(Point), compare);
72
73     int m = 1; // Initialize size of modified array
74     for (int i=1; i<n; i++)
75     {
76         while (i < n-1 && orientation(p0, points[i],
77             points[i+1]) == 0)
78             i++;
79         points[m] = points[i];
80         m++;
81     }

```



```

82     if (m < 3) return;
83
84     stack<Point> S;
85     S.push(points[0]);
86     S.push(points[1]);
87     S.push(points[2]);
88
89     for (int i = 3; i < m; i++)
90     {
91         while (orientation(nextToTop(S), S.top(),
92                             points[i]) != 2)
93             S.pop();
94         S.push(points[i]);
95     }
96     while (!S.empty()) // List Of The Points In The
97         Convex Hull
98     {
99         Point p = S.top();
100        cout << "(" << p.x << ", " << p.y << ")" <<
101            endl;
102        S.pop();
103    }
104
105 int main()
106 {
107     Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3, 1}, {3, 3}};

```

```

108     int n = sizeof(points)/sizeof(points[0]);
109     /* N is The Number of Points And points is
110     the list Of points */
111     convexHull(points, n);
112     return 0;
113 }

```

0.3.2 Shoelace Formula (python)

```

1 def PolygonArea(corners):
2     n = len(corners) # of corners
3     area = 0.0
4     for i in range(n):
5         j = (i + 1) % n
6         area += corners[i][0] * corners[j][1]
7         area -= corners[j][0] * corners[i][1]
8     area = abs(area) / 2.0
9     return area
10
11 # examples
12 corners = [(2.0, 1.0), (4.0, 5.0), (7.0, 8.0)]
13 print (PolygonArea(corners))

```

0.4 String

0.4.1 LCS

```

/* Dynamic Programming C/C++ implementation of LCS
problem */

```

```

2  #include<bits/stdc++.h>
3
4  int max(int a, int b)
5  {
6      return (a > b)? a : b;
7  }
8
9  int lcs( char *X, char *Y, int m, int n )
10 {
11     int L[m+1][n+1];
12     int i, j;
13
14     for (i=0; i<=m; i++)
15     {
16         for (j=0; j<=n; j++)
17         {
18             if (i == 0 || j == 0)
19                 L[i][j] = 0;
20
21             else if (X[i-1] == Y[j-1])
22                 L[i][j] = L[i-1][j-1] + 1;
23
24             else
25                 L[i][j] = max(L[i-1][j], L[i][j-1]);
26         }
27     }
28     return L[m][n];
29 }
30

```

```

31 int main()
32 {
33     char X[] = "AGGTAB";
34     char Y[] = "GXTXAYB";
35
36     int m = strlen(X);
37     int n = strlen(Y);
38
39     printf("Length of LCS is %dn", lcs( X, Y, m, n ) );
40
41     return 0;
42 }

```

0.4.2 LIS

```

1  typedef vector<int> VI;
2  typedef pair<int,int> PII;
3  typedef vector<PII> VPII;
4
5  #define STRICTLY_INCREASNG
6
7  VI LongestIncreasingSubsequence(VI v) {
8      VPII best;
9      VI dad(v.size(), -1);
10
11     for (int i = 0; i < v.size(); i++) {
12         #ifdef STRICTLY_INCREASNG
13             PII item = make_pair(v[i], 0);
14             VPII::iterator iter = lower_bound(best.begin(),

```

```

        best.end(), item);
15     item.second = i;
16 #else
17     PII item = make_pair(v[i], i);
18     VPII::iterator iter = upper_bound(best.begin(),
        best.end(), item);
19 #endif
20     if (iter == best.end()) {
21         dad[i] = (best.size() == 0 ? -1 : best.back().
            second);
22         best.push_back(item);
23     } else {
24         dad[i] = dad[iter->second];
25         *iter = item;
26     }
27 }
28
29 VI ret;
30 for (int i = best.back().second; i >= 0; i = dad[i
    ])
31     ret.push_back(v[i]);
32 reverse(ret.begin(), ret.end());
33 return ret;
34 }

```

0.4.3 KMP

```

1  /*
2  Finds all occurrences of the pattern string p within

```

```

        the
3  text string t. Running time is  $O(n + m)$ , where  $n$  and
        m
4  are the lengths of  $p$  and  $t$ , respectively.
5  */
6
7  #include <iostream>
8  #include <string>
9  #include <vector>
10
11 using namespace std;
12
13 typedef vector<int> VI;
14
15 void buildPi(string& p, VI& pi)
16 {
17     pi = VI(p.length());
18     int k = -2;
19     for(int i = 0; i < p.length(); i++) {
20         while(k >= -1 && p[k+1] != p[i])
21             k = (k == -1) ? -2 : pi[k];
22         pi[i] = ++k;
23     }
24 }
25
26 int KMP(string& t, string& p)
27 {
28     VI pi;
29     buildPi(p, pi);

```

```

30     int k = -1;
31     for(int i = 0; i < t.length(); i++) {
32         while(k >= -1 && p[k+1] != t[i])
33             k = (k == -1) ? -2 : pi[k];
34         k++;
35         if(k == p.length() - 1) {
36             // p matches t[i-m+1, ..., i]
37             cout << "matched at index " << i-k << ": ";
38             cout << t.substr(i-k, p.length()) << endl;
39             k = (k == -1) ? -2 : pi[k];
40         }
41     }
42     return 0;
43 }
44
45 int main()
46 {
47     string a = "AABAACAADAABAABA", b = "AABA";
48     KMP(a, b); // expected matches at: 0, 9, 12
49     return 0;
50 }

```

0.5 Other

0.5.1 Disjoint Set

```

1 // To represent Disjoint Sets
2 struct DisjointSets
3 {

```

```

4     int *parent, *rnk;
5     int n;
6
7     DisjointSets(int n)
8     {
9         this->n = n;
10        parent = new int[n+1];
11        rnk = new int[n+1];
12        for (int i = 0; i <= n; i++)
13        {
14            rnk[i] = 0;
15
16            parent[i] = i;
17        }
18    }
19
20    int find(int u)
21    {
22        if (u != parent[u])
23            parent[u] = find(parent[u]);
24        return parent[u];
25    }
26
27    void merge(int x, int y)
28    {
29        x = find(x), y = find(y);
30        if (rnk[x] > rnk[y])
31            parent[y] = x;
32        else
33            parent[x] = y;

```

```

33
34         if (rnk[x] == rnk[y])
35             rnk[y]++;
36     }
37 };

```

0.5.2 Date Transformation

```

1  //Dates (C++)
2
3  // Routines for performing computations on dates. In
   these routines,
4  // months are expressed as integers from 1 to 12,
   days are expressed
5  // as integers from 1 to 31, and years are expressed
   as 4-digit
6  // integers.
7
8  string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
9
10 // converts Gregorian date to integer (Julian day
   number)
11
12 int DateToInt (int m, int d, int y){
13     return
14         1461 * (y + 4800 + (m - 14) / 12) / 4 +
15         367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
16         3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +

```

```

17         d - 32075;
18 }
19
20 // converts integer (Julian day number) to Gregorian
   date: month/day/year
21
22 void IntToDate (int jd, int &m, int &d, int &y){
23     int x, n, i, j;
24
25     x = jd + 68569;
26     n = 4 * x / 146097;
27     x -= (146097 * n + 3) / 4;
28     i = (4000 * (x + 1)) / 1461001;
29     x -= 1461 * i / 4 - 31;
30     j = 80 * x / 2447;
31     d = x - 2447 * j / 80;
32     x = j / 11;
33     m = j + 2 - 12 * x;
34     y = 100 * (n - 49) + i + x;
35 }
36
37 // converts integer (Julian day number) to day of
   week
38
39 string IntToDay (int jd){
40     return dayOfWeek[jd % 7];
41 }

```

0.5.3 make CIN and COUT Run Faseter

```
1 ios_base::sync_with_stdio(false);
```