Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №4**

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3234

Баянов Р.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Яндекс контест

**Задача M. Цивилизация**

**Ход решения:**

Для решения данной задачи я использовал алгоритм Дейкстры. В самом начале подготавливаем данные, то есть создаём вектор строк, для того чтобы иметь какое-то представление карты игры в решении. Подготавливаем все структуры данных для алгоритма Дейкстры: создаём вектор предков *p*, создаём приоритетную очередь *q* и компаратор для неё, вектор посещённых вершин *visited* и вектор соседних вершин для выполнения релаксаций *vertices*. Затем по алгоритму мы будем ходить по нашей карте и аккуратно высчитывать наикратчайший путь для вершинки, в которую мы попали. И не забываем в конце каждого перехода, проводить релаксации, для подсчёта, возможно, нового изменившегося наикратчайшего пути для вершины. Сразу после того, как алгоритм Дейкстры закончит своё выполнение, по вектору предков *p* мы легко можем восстановить путь, проделанный до нашей целевой вершины. Выводим ответ в конце!

**Оценка сложности решения:**

Время: O ((V + E) log V)

Память: O (N \* M)

**Код решения:**

|  |
| --- |
| 1. #include <iostream> 2. #include <vector> 3. #include <string> 4. #include <queue> 5. #include <limits> 6. #include <algorithm> 8. enum Moves { 9. DOWN = 'S', 10. RIGHT = 'E', 11. LEFT = 'W', 12. UP = 'N' 13. }; 15. struct CompareSecond { 16. bool operator()(const std::pair<int, int>& a, const std::pair<int, int>& b) { 17. return a.second > b.second; 18. } 19. }; 21. int main() 22. { 23. std::cin.tie(0); 24. std::ios\_base::sync\_with\_stdio(0); 25. int INF = std::numeric\_limits<int>::max(); 26. int N, M, x1, y1, x2, y2; 27. std::cin >> N >> M >> x1 >> y1 >> x2 >> y2; 28. std::string moves = ""; 29. std::string str; 30. std::vector<std::string> grid(N); 31. int start = (x1 - 1) \* M + (y1 - 1); 32. int goal = (x2 - 1) \* M + (y2 - 1); 33. std::vector<int> visited(N \* M, INF); 34. visited[start] = 0; 35. for (int i = 0; i < N; i++) { 36. std::cin >> grid[i]; 37. } 38. std::vector<int> p (N \* M); 39. std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, CompareSecond> q; 40. q.push({start, 0}); 41. std::vector<std::pair<int, int>> vertices; 42. while (!q.empty()) { 43. std::pair<int, int> v = q.top(); 44. x1 = v.first / M; 45. y1 = v.first % M; 46. q.pop(); 47. if (x1 == x2 && y1 == y2) { 48. break; 49. } 50. if (grid[x1][y1] == '#') { 51. continue; 52. } 53. if (x1 - 1 >= 0 && grid[x1 - 1][y1] != '#') { 54. vertices.push\_back({ v.first - M, (grid[x1 - 1][y1] == 'W') ? 2 : 1 }); 55. } 56. if (x1 + 1 < N && grid[x1 + 1][y1] != '#') { 57. vertices.push\_back({ v.first + M, (grid[x1 + 1][y1] == 'W') ? 2 : 1 }); 58. } 59. if (y1 - 1 >= 0 && grid[x1][y1 - 1] != '#') { 60. vertices.push\_back({ v.first - 1, (grid[x1][y1 - 1] == 'W') ? 2 : 1 }); 61. } 62. if (y1 + 1 < M && grid[x1][y1 + 1] != '#') { 63. vertices.push\_back({ v.first + 1, (grid[x1][y1 + 1] == 'W') ? 2 : 1 }); 64. } 65. for (std::pair<int, int> val : vertices) { 66. int new\_len = val.second + v.second; 67. if (visited[val.first] == INF || visited[val.first] > new\_len) { 68. q.push({val.first, new\_len}); 69. visited[val.first] = new\_len; 70. p[val.first] = v.first; 71. } 72. } 73. vertices.clear(); 74. } 75. if (visited[goal] == INF) { 76. std::cout << -1; 77. return 0; 78. } 79. if (visited[goal] == 0) { 80. std::cout << visited[goal]; 81. } 82. else { 83. std::cout << visited[goal] << std::endl; 84. } 85. for (int v = goal; v != start; v = p[v]) { 86. p.push\_back(v); 87. } 88. p.push\_back(start); 89. std::reverse(p.begin(), p.end()); 90. for (int i = 0; i < N \* M; i++) { 91. if (p[i] == goal) { 92. break; 93. } 94. int diff = p[i + 1] - p[i]; 95. if (diff == -1) { 96. moves += LEFT; 97. } 98. else if (diff == 1) { 99. moves += RIGHT; 100. } 101. else if (diff == M) { 102. moves += DOWN; 103. } 104. else if (diff == -M) { 105. moves += UP; 106. } 107. } 108. std::cout << moves; 109. return 0; 110. } |

**Задача N. Свинки-копилки**

**Ход решения:**

Для решения данной задачи будем использовать алгоритм dfs. Чтобы получить ответ нам нужно посчитать все компоненты связности нашего графа. При обработке входных данных будем каждое ребро дублировать, чтобы при попадании в какую-то компоненту связности мы могли из любой её вершины обойти все вершины этой компоненты. Мы знаем, что dfs обойдёт все вершины, поэтому при остановке алгоритма dfs мы можем с уверенностью сказать, что одна компонента связности обнаружена и все её вершины помечены. Затем, мы будем искать другую непомеченную вершину и снова запустим dfs, но уже с этой вершины. Таким образом после каждого dfs увеличиваем счётчик и получаем ответ.

**Оценка сложности решения:**

Время: O (V + E) = O (N)

Память: O (N2)

**Код решения:**

|  |
| --- |
| 1. #include <iostream> 2. #include <vector> 3. #include <queue> 5. enum Color { 6. WHITE = 0, 7. BLACK = 1 8. }; 10. void dfs( std::vector<std::vector<int>>& graph, std::vector<int>& visited, int v) { 11. visited[v] = BLACK; 12. for (int u : graph[v]) { 13. if (visited[u] == WHITE) { 14. dfs(graph, visited, u); 15. } 16. } 17. } 19. int main() 20. { 21. int n; 22. std::cin >> n; 23. std::vector<std::vector<int>> graph(n + 1); 24. std::vector<int> visited(n + 1, WHITE); 25. int e; 26. for (int v = 1; v < n + 1; ++v) { 27. std::cin >> e; 28. graph[v].push\_back(e); 29. graph[e].push\_back(v); 30. } 31. int count = 0; 32. for (int i = 1; i < graph.size(); i++) { 33. if (visited[i] == WHITE) { 34. count++; 35. dfs(graph, visited, i); 36. } 37. } 38. std::cout << count; 39. return 0; 40. } |

**Задача O. Долой списывание!**

**Ход решения:**

Для решения данной задачи я использовал алгоритм bfs. Нетрудно догадаться, что здесь надо проверить является ли граф двудольным. Таким образом, если граф двудольный мы легко сможешь разбить лкшат на две группы. В этом нам поможет алгоритм bfs. Мы будем аккуратно обходить граф в ширину и раскрашивать вершины в два цвета. И если мы всё-таки встретим ситуацию, когда две соседние вершины раскрашены в один и тот же цвет, то граф не двудольный. Вот и всё!

**Оценка сложности решения:**

Время: O (V + E) = O (N)

Память: O (N2)

**Код решения:**

|  |
| --- |
| #include <iostream>  #include <vector>  #include <queue>  enum Color {  WHITE,  BLUE,  RED  };  #define WHITE -1  #define BLUE 0  #define RED 1  bool is\_bipartite(std::vector<std::vector<int>>& graph) {  std::queue<int> q;  std::vector<int> visited(graph.size(), WHITE);  for (int s = 1; s < graph.size(); s++) {  if (visited[s] == WHITE) {  visited[s] = BLUE;  q.push(s);  while (!q.empty()) {  int u = q.front();  q.pop();  for (int v : graph[u]) {  if (visited[v] == -1) {  visited[v] = 1 - visited[u];  q.push(v);  }  else if (visited[u] == visited[v]) {  return false;  }  }  }  }  }  return true;  }  int main()  {  int N, M;  std::cin >> N >> M;  std::vector<std::vector<int>> graph(N + 1);  int v, u;  for (int i = 0; i < M; i++) {  std::cin >> v >> u;  graph[u].push\_back(v);  graph[v].push\_back(u);  }  if (is\_bipartite(graph)) {  std::cout << "YES";  }  else {  std::cout << "NO";  }  return 0;  } |

**Задача P. Авиаперелёты**

**Ход решения:**

Для решения данной задачи я использовал dfs и бин поиск. Грубо говоря, мы найдём в матрице максимум, который потом будет служить нам верхней границей для диапазона бин поиска. Мы создадим функцию проверки графа на то, что с каким-то определённым размером бака самолёта, мы сможем облететь все города. В этой функции проверки будем проходиться двумя dfs по графу в прямом направлении и в обратном. Если всё же при данный размер бака нам подходит, то самолёт облетит все города и все вершины графа станут помеченными и мы точно будем знать подходит нам такой бак или нет. И так далее пока бин поиск не обнаружит минимальный такой бак. Готово!

**Оценка сложности решения:**

Время: O (N2 log E)

Память: O(V), где V – кол-во вершин

**Код решения:**

|  |
| --- |
| 1. #include <iostream> 2. #include <vector> 3. #include <limits.h> 5. struct vertex { 6. int id; 7. int weight; 8. };  11. void dfs1(std::vector<std::vector<vertex>>& graph, int val, int v, std::vector<int>& visited) { 12. visited[v] = 1; 13. for (int i = 0; i < graph.size(); ++i) { 14. if (!visited[i] && graph[i][v].weight <= val) { 15. dfs1(graph, val, i, visited); 16. } 17. } 18. } 20. void dfs2(std::vector<std::vector<vertex>>& graph, int val, int v, std::vector<int>& visited) { 21. visited[v] = 1; 22. for (int i = 0; i < graph.size(); ++i) { 23. if (!visited[i] && graph[v][i].weight <= val) { 24. dfs2(graph, val, i, visited); 25. } 26. } 27. } 29. bool check(std::vector<std::vector<vertex>>& graph, int val) { 30. int size = graph.size(); 31. std::vector<int> visited (size, 0); 32. dfs1(graph, val, 0, visited); 33. for (int i = 0; i < size; ++i) { 34. if (!visited[i]) { 35. return false; 36. } 37. } 38. std::fill(visited.begin(), visited.end(), 0); 39. dfs2(graph, val, 0, visited); 40. for (int i = 0; i < size; ++i) { 41. if (!visited[i]) { 42. return false; 43. } 44. } 45. return true; 46. } 48. int bin\_search(int max, std::vector<std::vector<vertex>>& graph) { 49. int l = 0; 50. int r = max; 51. while (r - l > 1) { 52. int m = (l + r) / 2; 53. if (!check(graph, m)) { 54. l = m; 55. } 56. else { 57. r = m; 58. } 59. } 60. return r; 61. } 63. int main() 64. { 65. int n; 66. std::cin >> n; 67. vertex v; 68. std::vector<std::vector<vertex>> graph(n); 69. int oil; 70. int max = INT\_MIN; 71. for (int i = 0; i < n; i++) { 72. for (int j = 0; j < n; j++) { 73. std::cin >> oil; 74. max = std::max(oil, max); 75. vertex v; 76. v.id = j; 77. v.weight = oil; 78. graph[i].push\_back(v); 79. } 80. } 81. std::cout << bin\_search(max, graph); 82. return 0; 83. } |

Тимус

**Задача 1162. Currency Exchange**

**Ход решения:**

Для каждого ребра в графе будем проверять можно ли увеличить кол-во денег в вершине, в которую ведёт это ребро, обменивая валюту в вершине, из которой ведёт это ребро. Если да, то сума денег в вершине обновляется. Затем проверка на то, что у нас есть цикл в графе, который позволяет увеличить свою сумму денег. И выводим ответ на вопрос!

**Оценка сложности решения:**

Время: O (V \* E)

Память: O (V + E)

**Код решения:**

|  |
| --- |
| 1. #include <iostream> 2. #include <list> 3. #include <vector> 4. using namespace std; 6. struct edge { 7. int from, to; 8. double rate, commission; 9. }; 11. int main() { 12. int N, M, S, A, B; 13. double V, RAB, CAB, RBA, CBA; 14. cin >> N >> M >> S >> V; 15. list<edge> edges; 16. S--; 17. for (int i = 0; i < M; i++) { 18. cin >> A >> B >> RAB >> CAB >> RBA >> CBA; 19. A--; B--; 20. edges.push\_back({ A, B, RAB, CAB }); 21. edges.push\_back({ B, A, RBA, CBA }); 22. } 24. std::vector<double> currency(N, 0); 25. currency[S] = V; 27. for (int i = 0; i < N - 1; i++) 28. for (edge x : edges) 29. if ((currency[x.from] - x.commission) \* x.rate > currency[x.to]) 30. currency[x.to] = (currency[x.from] - x.commission) \* x.rate; 32. for (edge x : edges) 33. if ((currency[x.from] - x.commission) \* x.rate > currency[x.to]) { 34. cout << "YES"; 35. return 0; 36. } 37. cout << "NO"; 38. return 0; 39. } |

**Задача 1450. Российские газопроводы**

**Ход решения:**

Для решения этой задачи создадим вектор d, который будет хранить расстояния от начальной вершины до всех остальных вершин в графе. Затем в цикле, выполняется алгоритм Беллмана-Форда. На каждой итерации будем проверять для каждого ребра в графе можно ли улучшить расстояние до вершины, в которую ведёт это ребро ,используя текущее расстояние до вершины, из которой ведёт это ребро и вес этого ребра. В конце проверяем найдено ли решение. Если расстояние до конечной вершины равно 1, то решение не найдено, в противном случае выводим ответ. Всё!

**Оценка сложности решения:**

Время: O (N \* M)

Память: O (N + M)

**Код решения:**

|  |
| --- |
| 1. #include <iostream> 2. #include <vector> 4. using namespace std;  7. int main() { 9. ios::sync\_with\_stdio(0); 10. cin.tie(0); 11. cout.tie(0); 13. int n, m, start, finish; 14. cin >> n >> m; 16. vector<vector<int> > edges(m, vector<int>(3)); 17. for (int i = 0; i < m; ++i) { 18. cin >> edges[i][0] >> edges[i][1] >> edges[i][2]; 19. edges[i][0]--; 20. edges[i][1]--; 21. } 22. cin >> start >> finish; 23. vector<int> d(n, -1); 25. d[--start] = 0; 26. for (int i = 0; i < n - 1; ++i){ 27. for (int j = 0; j < m; ++j){ 28. if (d[edges[j][0]] > -1) { 29. d[edges[j][1]] = max(d[edges[j][1]], d[edges[j][0]] + edges[j][2]); 30. } 31. } 32. } 33. if(d[--finish] == 1) { 34. cout << "No solution"; 35. return 0; 36. } 37. cout << d[finish]; 39. return 0; 40. } |