Renat Norderhaug

CS 477

HW #6

11/14/19

1.

a) Write a recursive formula for an optimal solution

We are trying to maximize a subset S, which contains a set of observable events. The solution to computing it dynamically is to find the shortest distance between all the points, so the Dj of each coordinate subtracted with the Dj of every other coordinate. Then to pick the lowest values. The last event must be visited.

$D_{k,i,j} = \max \{ D_{k-1,i,j} \text{ or } D_{k-1,i,k} + D_{k-1,k,j} \}$

Here we have two vertices i, and j and k represents the intermediary nodes. We want to find the max path which fits the last node as well as has the highest number of K values.

b) Write an algorithm that computes the optimal value to this problem based on the recurrence above

```
// topological sorting for the events and there coordinates
void Graph::longestPath(int s)
{
    stack<int> Stack;
    int dist[V];

    // Mark all the vertices as not visited
    bool* visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to store Topological
    // Sort starting from all vertices one by one
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    // Initialize distances to all vertices as infinite and
    // distance to source as 0
    for (int i = 0; i < V; i++)
        dist[i] = NINF;
    dist[s] = 0;

    // Process vertices in topological order
    while (Stack.empty() == false) {
        // Get the next vertex from topological order
        int u = Stack.top();
        Stack.pop();

        // Update distances of all adjacent vertices
        list<AdjListNode>::iterator i;
        if (dist[u] != NINF) {
            for (i = adj[u].begin(); i != adj[u].end(); ++i)
                if (dist[i->getV()] < dist[u] + i->getWeight())
                    dist[i->getV()] = dist[u] + i->getWeight();
        }
    }

    // Print the calculated longest distances
    for (int i = 0; i < V; i++)
        (dist[i] == NINF) ? cout << "INF " : cout << dist[i] << " ";
```

```
rnorderhaug@ecc-d-03:~/Desktop/CS477/hw6$ ./hw6
Following are the events you could make
INF 0 2 9 8 10
PROBLEM B:
sub-problem 1: max between 21963+5 and 21963 is 21968
sub-problem 2: max between 21963+6 and 21968 is 21969
sub-problem 3: max between 21968+5 and 21969 is 21973
sub-problem 4: max between 21969+1 and 21973 is 21973
OPTIMAL SOLUTION: 21973

PROBLEM C:
OPTIMUM VALUE: 21973 is the maximum event following the below table:
                                        Coordinates

         |   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
18  19  20  |
```

c) Update the algorithm in part B to enable the reconstruction of the optimal solution

```
// global variables
int storedVal[20];
int placed[20];
int removed[20];


int optVal(int D, int n, int x[], int r[]){

        // must be 5 miles apart
        int distance = 5;

        // array for storing max revenue at every mile
        int opt[D + 1];
        memset(opt, 0, sizeof(opt));

        // incremental variables
        int nextBoard = 0;
        int subProb = 1;

        for(int i = 0; i < D + 1; i++){

                if(nextBoard < n){
                        if(x[nextBoard] != i){
                                opt[i] = opt[i - 1];
                        } // end if

                        else{
                                if(i <= distance){
                                        opt[i] = max(opt[i - 1], r[nextBoard]);

                                        // problem b
                                        cout << "sub-problem " << subProb << ": max between " << opt[i - 1] - 1 << "
and " << r[nextBoard] - 1 << " is " << opt[i] - 1 << endl;
                                        subProb++;

                                } // end if
                                else{
                                        opt[i] = max(opt[i - distance - 1] + r[nextBoard], opt[i - 1]);

                                        // problem b
                                        cout << "sub-problem " << subProb << ": max between " << opt[i - distance - 1]
- 1 << "+" << r[nextBoard] << " and " << opt[i - 1] - 1 << " is " << opt[i] - 1 << endl;
                                        subProb++;

                                        // table/value reference
```

```
PROBLEM C:
OPTIMUM VALUE: 21973 is the maximum event following the below table:
                                        Coordinates

         |   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
18  19  20  |

  PLACED|   0   0   0   0   0   1   1   0   0   0   0   1   0   1   0   0   0
0   0   0  |
 REMOVED|   1   1   0   0   0   0   1   0   0   0   0   0   0   1   0   0   0
0   0   0  |
  EVENTS|  21963  21963  21963  21963  21963  21968  21969  21969  21969  21969
 21969  21973  21973  21973  21973  21973  21973  21973  21973  21973  |
```

d) Write an algorithm that output the events you choose to view in the optimal solution.

```
// topological sorting for the events and there coordinates
void Graph::longestPath(int s)
{
    stack<int> Stack;
    int dist[V];

    // Mark all the vertices as not visited
    bool* visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to store Topological
    // Sort starting from all vertices one by one
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    // Initialize distances to all vertices as infinite and
    // distance to source as 0
    for (int i = 0; i < V; i++)
        dist[i] = NINF;
    dist[s] = 0;

    // Process vertices in topological order
    while (Stack.empty() == false) {
        // Get the next vertex from topological order
        int u = Stack.top();
        Stack.pop();

        // Update distances of all adjacent vertices
        list<AdjListNode>::iterator i;
        if (dist[u] != NINF) {
            for (i = adj[u].begin(); i != adj[u].end(); ++i)
                if (dist[i->getV()] < dist[u] + i->getWeight())
                    dist[i->getV()] = dist[u] + i->getWeight();
        }
    }

    // Print the calculated longest distances
    for (int i = 0; i < V; i++)
        (dist[i] == NINF) ? cout << "INF " : cout << dist[i] << " ";
```

```
Terminal
File  Edit  View  Search  Terminal  Help
Following are the events you could make
INF 0 2 9 8 10
```

This last screenshot is ran on a x,y matrix with events and coordinates.

6. Matrix multiplication follows the associative property, which is a(b*c) = (a*b)*c

For the Given matrix's A, B, C , D and E we have 10 different orderings:

- **((AB)C(DE))**
- **(A(BC)DE)**
- **((AB)(CD)E)**
- **(A(B(CD)E))**
- **((AB)(C(DE)))**
- **(A(B(C(DE))))**
- **(A(BC)(DE))**
- **((((AB)C)D)E)**
- **(A(B(CD)E))**
- **(((AB)C)(DE))**