1. Pearson Test and Contingency table

2. 17+3 =20 (有 2k 个奇顶点，那么它可以用 k 笔画成，并且至少要用 k 笔画成)

3. Can only work out using density function…

4.

(1) Stop. since 35 > 1/6\*(0+37+38+39+40+41) = 32.5

(2) 46

(3) before you reach 18, you should continue

5. 1/9, 5/18, 11/18

6.

(1) 1

(2) (N-1)/N + 2 \* N(N-1)/2 \* 1/(N\*N\*(N-1)) = 1

(3) N

(4) (N+1)\*N/2

(5) (N-1)\*N/2

7. 0.2 to 0.3

8. break 1/N into binary representation, define event accord to the 1 and 0

9. How to define “miraculously fixed” and how many bridges are weak?

10. The function fun()’s input is a reference on a constant pointer which point to the constant integer, and fun() returns a constant pointer to a constant integer, without modifying the non-mutable members of its class

11.

/\*

\* Delete element at a given position

\*/

void single\_llist::delete\_pos()

{

int pos, i, counter = 0;

if (start == NULL)

{

cout<<"List is empty"<<endl;

return;

}

cout<<"Enter the position of value to be deleted: ";

cin>>pos;

struct node \*s, \*ptr;

s = start;

if (pos == 1)

{

start = s->next;

}

else

{

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos > 0 && pos <= counter)

{

s = start;

for (i = 1;i < pos;i++)

{

ptr = s;

s = s->next;

}

ptr->next = s->next;

}

else

{

cout<<"Position out of range"<<endl;

}

free(s);

cout<<"Element Deleted"<<endl;

}

}

12.

#ifndef \_MATRIX\_H

#define \_MATRIX\_H

#include <vector>

template <typename T> class Matrix {

private:

std::vector<std::vector<T> > mat;

unsigned rows;

unsigned cols;

public:

Matrix(unsigned \_rows, unsigned \_cols, const T& \_initial);

Matrix(const Matrix<T>& rhs);

virtual ~Matrix();

// Operator overloading, for "standard" mathematical matrix operations

Matrix<T>& operator=(const Matrix<T>& rhs);

// Matrix mathematical operations

Matrix<T> operator+(const Matrix<T>& rhs);

Matrix<T>& operator+=(const Matrix<T>& rhs);

Matrix<T> operator-(const Matrix<T>& rhs);

Matrix<T>& operator-=(const Matrix<T>& rhs);

Matrix<T> operator\*(const Matrix<T>& rhs);

Matrix<T>& operator\*=(const Matrix<T>& rhs);

Matrix<T> transpose();

// Matrix/scalar operations

Matrix<T> operator+(const T& rhs);

Matrix<T> operator-(const T& rhs);

Matrix<T> operator\*(const T& rhs);

Matrix<T> operator/(const T& rhs);

// Matrix/vector operations

std::vector<T> operator\*(const std::vector<T>& rhs);

std::vector<T> diag\_vec();

// Access the individual elements

T& operator()(const unsigned& row, const unsigned& col);

const T& operator()(const unsigned& row, const unsigned& col) const;

// Access the row and column sizes

unsigned get\_rows() const;

unsigned get\_cols() const;

};

#endif

13. No.

14. Yes, doable.

15. Manacher’s Algorithm (difficult to understand…) O(N) time

Easy way O(N^2) time and O(1) space:

string expandAroundCenter(string s, int c1, int c2) {

int l = c1, r = c2;

int n = s.length();

while (l >= 0 && r <= n-1 && s[l] == s[r]) {

l--;

r++;

}

return s.substr(l+1, r-l-1);

}

string longestPalindromeSimple(string s) {

int n = s.length();

if (n == 0) return "";

string longest = s.substr(0, 1); // a single char itself is a palindrome

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

string p1 = expandAroundCenter(s, i, i);

if (p1.length() > longest.length())

longest = p1;

string p2 = expandAroundCenter(s, i, i+1);

if (p2.length() > longest.length())

longest = p2;

}

return longest;

}

16. Two pass: Reverse the entire string, then reverse the letters of each individual word.

Or in python use split()…

17. Idea:

1) Get the maximum profit with one transaction as in quiz 1. Keep down the start and end position

2) The start and end positions will be included in the result of two transactions. Two possibilities:

A) it is one full transaction

B) belong to two separate transactions, start is the first transaction’s start, end is second transaction’s end

3) for A), get the maximum profit in [0, start) and maximum profit in (end, N], take the larger to be second transaction

For B), get the maximum profit with one transaction within (start, end) but **in reverse order**, say (newstart, newend), then the two transaction would be [start, newstart], [newend, end]

4) take the maximum of A) and B)

int maxProfit(vector<int>& prices, int start, int end, int dir = 1) {

int res = 0, mini = prices[start];

int i = start;

while (i != end) {

if (prices[i] < mini) {

mini = prices[i];

} else {

if (prices[i] - mini > res) res = prices[i] - mini;

}

i += dir;

}

return res;

}

int maxProfit(vector<int>& prices) {//secret lay in the max and min prices

int n = prices.size();

if (n == 0) return 0;

int minind = 0, tmpind = 0, maxind = 0, res = 0, tmp;

int mini = prices[0];

for (int i = 1; i < n; i++) {

if (prices[i] <= mini) {

mini = prices[i];

tmpind = i;

}

else {

tmp = prices[i] - mini;

if (tmp > res) {

res = tmp;

maxind = i;

minind = tmpind;

}

}

}

int m = 0, m1, m2;

if (minind > 1) m = maxProfit(prices, 0, minind);

m1 = res + m;

if (maxind < n - 2) m = maxProfit(prices, maxind + 1, n);

m2 = res + m;

if (minind < maxind - 2) res += maxProfit(prices, maxind - 1, minind, -1);

return max(max(m1, m2), res);

}

18. ACM question